UNIVERSITÀ DEGLI STUDI DI NAPOLI

"FEDERICO II"

Scuola Politecnica e delle Scienze di Base

Area Didattica di Scienze Matematiche Fisiche e Naturali

Dipartimento di Fisica "Ettore Pancini"

Laurea Magistrale in Fisica

Fuzzy LTL Games

Relatori:

Prof. Giovanni Acampora

Prof. Aniello Murano

Candidato:

Andrea Colecchia

Matr. $N94000215$

Anno Accademico 2019/2020

Preface

In the past decades we have witnessed an increase of interest towards two different aspects of systems' theory. On one side, the necessity to be able to specify and to check properties of systems has led to the introduction of temporal logics, such as LTL (Linear-time Temporal Logic), and to the study of related problems, such as model checking or LTL games. On the other side, the persistent will of finding new and better ways to control systems has enlarged the already considerable family of controllers available: in particular, this search has conducted to the introduction of fuzzy controllers. The present work aims at building a bridge between these apparently separate sectors of systems' theory. Our goal is to show how the theoretical tools used to check properties, and those applied to individuate strategies to assure they are fulfilled, can be used to model the situation to be controlled, and therefore to instruct the controller itself. Having chosen to focus our attention on fuzzy controllers, as a new and highly promising kind of control system, we first had to extend the usual LTL techniques to allow fuzzy values for the variables. Besides, the strategies emerging from our approach prove different from the fuzzy controllers usually defined. The main body of this thesis therefore consists in defining and solving fuzzy LTL games, and in showing how to transform them in fuzzy controllers. Bibliographical remarks and consideration about the algorithms developed complete the work.

Contents

Preface **i**

1 Basic notions of control **4**

1.1 Introduction 4

1.1.1 Fuzzy logic 7

1.1.2 Fuzzy sets 9

1.2 Fuzzy control systems 10

1.2.1 Present fuzzy controllers' limitations and our proposal . . 19

2 Reactive systems and Linear-time Temporal Logic (LTL) **22**

2.1 Reactive systems 22

2.1.1 Formal definition of Kripke structures 23

2.1.2 Usefulness of Kripke structures in modeling reactive systems 24

2.1.3 Example of a reactive system described by a Kripke structure 24

2.2 Linear-time Temporal Logic (LTL) 25

2.2.1 Formal definition of LTL 25

2.2.2 Usefulness of LTL in modeling specifications of reactive systems 28

2.2.3 Fuzzification of LTL 29

3 Fuzzy LTL games **31**

3.1 Basic notions of automata theory 31

3.1.1 Formal definitions 32

3.1.2 Automata problems of interest in computer science 34

3.2 Model-checking problem for LTL specifications 35

3.2.1 Formal statement of the model-checking problem 35

3.2.2 Solving the model-checking problem using the Vardi-Wolper procedure 35

3.3 LTL games 36

3.3.1 Formal definitions of LTL, Büchi and parity games 36

3.3.2 Solving LTL games 38

3.4 Fuzzy LTL games 39

3.4.1 Formal definition of fuzzy LTL games 39

3.4.2 Solving fuzzy LTL games 40

4 LTL[Z] controller synthesis **44**

4.1 Construction of an LTL[Z]-based controller 44

4.2 An example 46

5 Fragments of LTL **49**

5.1 Difficulties in the determinization of a Büchi automaton 50

5.2 Fragments of LTL 50

5.3 Fragments of LTL[Z] 51

Conclusions **55**

Brief survey 55

Possible future developments 56

Further readings **57**

Other attempts to fuzzify LTL 57

System-environment models using games different from LTL 58

A Constructing the Vardi-Wolper Büchi automaton **60**

Chapter 1

Basic notions of control

This chapter starts with a brief introduction to control systems in general, and proceeds with a detailed discussion of fuzzy control systems.

1.1 Introduction

Generally speaking, a system can be defined as a well-delimited portion of an environment. According to the case, we may speak of mechanical systems, thermodynamic systems, chemical systems, etc. A control system is a device able to enforce a desired behavior of a given system. This definition may look rather vague, but vagueness is unavoidable due to the large number of different control systems. A brief historical overview, as provided for example in [1], will probably provide a useful introduction. Control systems were born in the industrial machines. Many historians mark the beginning of the industrial revolution with James Watt's steam engine (1769), and Watt provided it with a "governor", whose aim was to prevent oscillations in the steam flow (Watt's governor is depicted in

Figure 1.1)[1].

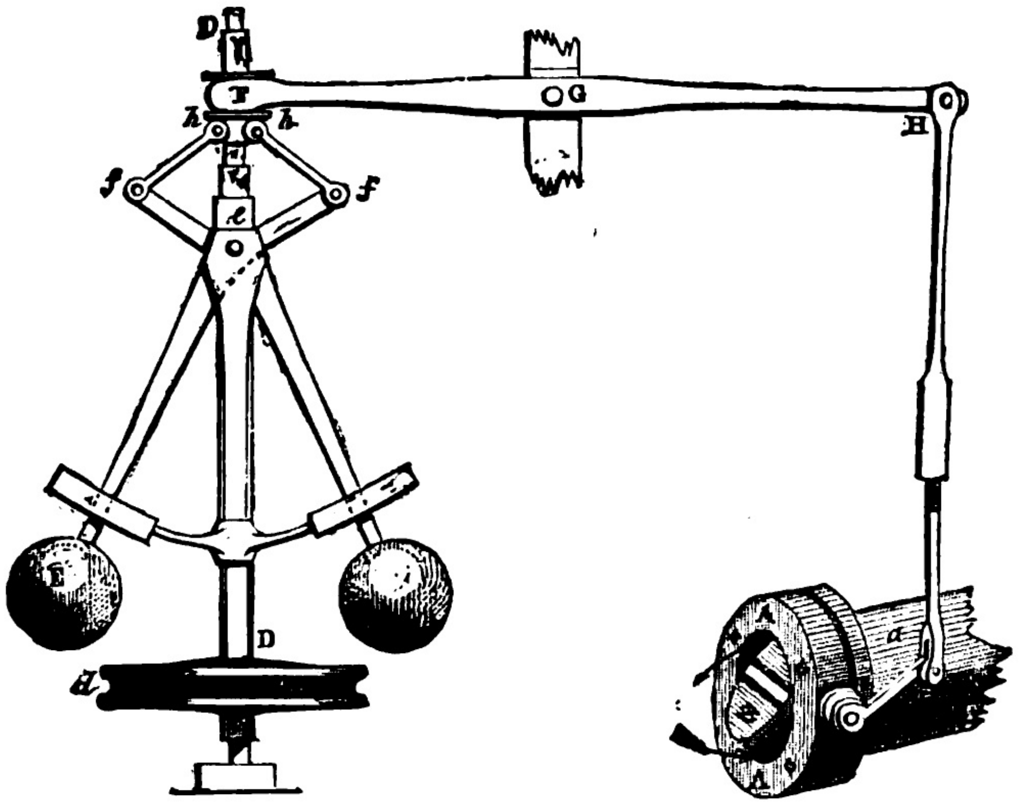


Figure 1.1: Watt's governor as represented in *Discoveries & Inventions of the Nineteenth Century*, by R. Routledge.

Watt's governor can therefore be considered the first control system. It was a mechanical device consisting of a conical pendulum whose rotating speed depended on the speed of the steam flow. At the same time, the pendulum masses were connected to a valve capable of reducing the fuel income of the furnace producing steam. The faster the pendulum rotated, the more the masses went upward because of centrifugal force and the more the valve was occluded. This

[1]To be precise, Watt based the governor on a very similar device invented by Christiaan Huygens to regulate the distance and the pressure between millstones in windmills. Watt himself never claimed to have invented the device. Nonetheless, its application to steam engines is widely recognized as the real starting point of control theory.

mechanism then prevented the steam from flowing too fast. For many years, no substantial progress was made in the field. It was only in 1868 that James Clerk Maxwell firstly provided a mathematical model for a governor controlling a steam engine in [2]. Maxwell described different types of governors using differential equations. Therefore, his approach can be called analytic: a mathematical model of the system to be controlled, and of its interaction with the controlling system is given, and is used to implement the controller itself. This framework was very fruitful, and has been improved by many other scientists in the following years. In the twentieth century it led to feedback control systems, where the actual output is compared with the desired one, and the controller works to keep the difference as small as possible. However, in the last decades, a new approach was developed. In his 1965 paper [3] Lotfi A. Zadeh introduced the expression *fuzzy logic* to indicate a many-valued logic where a sentence, besides the ordinary boolean values of *0* (*False*) and *1* (*True*), can assume any value between these, enabling one to speak about partial truth in a quantitative way. Accordingly, he described a new set theory, which he called fuzzy set theory, where the concept of an element belonging to a set was not crisp: an element could belong to the set, not belong, or have a partial degree of belonging expressed by a number comprised between *0* and *1*.[2] In the following decades fuzzy logic was applied for the first time to industry, especially in Japan: Hitachi showed its feasibility for controlling trains in a subway network; Mitsubishi used it to design an air conditioner; Matsuhita employed it in the construction of a vacuum cleaner, and so on (see [4] [5] [6]). Fuzzy control theory was born. Control systems in general implement some devices to measure the variables they work with (these devices are called sensors), and some

[2]It is important to underline that Zadeh just introduced the term “fuzzy” and the concept of fuzzy set; many-valued logics had already been studied since the 1920s, notably by Łukasiewicz and Tarski.

others to influence the value of these variables (these are called actuators). Fuzzy controllers collect inputs from sensors, map them into appropriate fuzzy sets, use rules to produce fuzzy outputs for the actuators and then map back these outputs into settings for the actuators. This approach is quite different from the one originated from Maxwell's paper. In the following we will only deal with those types of control systems. Before we define fuzzy control systems, an introduction to fuzzy logic and fuzzy sets is needed.

1.1.1 Fuzzy logic

In fuzzy logic we suppose that sentences' truth values can be any real number in the interval $[0,1]$. Accordingly, any logical operator has to be able to deal with fuzzy truth values. If we want to keep using the boolean operators $\wedge$ (whose intuitive meaning is that of the conjunction "and"), $\vee$ (which expresses disjunction, as the word "or"), $\neg$ (representing negation, like particle "not") and $\rightarrow$ ("implication", in the sense that the left side implies the right one), for example, we have to extend their usual definitions to enable them to deal with fuzzy truth values. We recall that, in boolean logic, these operators' action is usually described using the following tables:

Table 1.1: *AND table*

p	q	$p \wedge q$
True	*True*	*True*
True	*False*	*False*
False	*True*	*False*
False	*False*	*False*

Table 1.2: *OR table*

p	q	$p \vee q$
True	*True*	*True*
True	*False*	*True*
False	*True*	*True*
False	*False*	*False*

Table 1.3: *NOT table*

p	$\neg p$
True	*False*
False	*True*

Table 1.4: *IMPLICATION table*

p	q	$p \rightarrow q$
True	*True*	*True*
True	*False*	*False*
False	*True*	*True*
False	*False*	*True*

Tables can be used in ordinary boolean logic because sentences can only take a finite amount of values. When infinitely many values are allowed, the rules which fix the semantics of these operators must take another form. Usually, some useful properties are required for such rules. First of all, it is required that they reproduce boolean operators when acting on formulas having the boolean values *True* and *False*, i.e., *1* and *0*. Also, it is typically required that conjunction be commutative, monotonic, associative and admit *1* as identity element. Many non-equivalent operators satisfying these properties have been proposed. In the following we will consider the form proposed by Zadeh:

$$\neg x = 1 - x; \tag{1.1}$$

$$x \vee y = max\left\{x, y\right\}; \tag{1.2}$$

$$x \wedge y = min\left\{x, y\right\}; \tag{1.3}$$

$$x \to y = max\left\{min\left\{x, y\right\}, 1 - y\right\}. \tag{1.4}$$

It can be easily verified that they fulfill the aforementioned properties and satisfy De Morgan's laws. With their definition boolean logic is successfully fuzzified. Of course, if one considers additional boolean operators, additional rules must be given for them.

1.1.2 Fuzzy sets

While in classic set theory, given a set A and an element p, the only possibilities are "p belongs to A" and "p does not belong to A", in fuzzy set theory an element can belong to a set with any degree in range $[0, 1]$. In order to evaluate what is the degree of belonging of an element to a set, membership functions are needed. Let's explain this point with an example. Suppose we have the fuzzy sets *"Short people"* and *"Tall people"*. Each of them needs a membership function to assign degrees of belonging to any possible element, i.e., to each person. Possible membership functions, $Short(h)$ and $Tall(h)$, where h indicates the height of the person whose degrees of belonging we wish to find, may be

$$Short(h) = \begin{cases} 1\ for\ 0.20m \leq h \leq 1m \\ -1.428h + 2.4\ for\ 1m \leq h \leq 1.70m, \\ 0\ for\ 1.70m \leq h \end{cases} \tag{1.5}$$

and

$$Tall(h) = \begin{cases} 0 \; for \; 0.20m \leq h \leq 1m \\ 1.428h - 1.428 \; for \; 1m \leq h \leq 1.70m. \\ 1 \; for \; 1.70m \leq h \end{cases} \tag{1.6}$$

Using these membership functions a person whose height is 1.64m would belong to the set *"Short people"* with degree 0.059 and would belong to the set *"Tall people"* with degree 0.913. Of course, other membership functions are possible, leading to different degrees of belonging: their choice is arbitrary as that of the number of fuzzy sets to be used.

1.2 Fuzzy control systems

In Section 1.1 we said that the first studies in control theory used differential equation to model the system to be controlled and its interaction with the controller. In fuzzy control theory, instead, we adopt a more empirical point of view. As underlined in works like [7], we suppose we can talk with an expert who can successfully control the actuator. Typically, such experts do not have a mathematical understanding of the phenomena they manage to control, rather, they have an instinctive one: think for example of automobile drivers or football players. Usually they have some "rules", rather than precise mathematical equations, which prescribe what to do in any possible situation. A fuzzy controller, too, uses fuzzy *if-then* rules to relate variables' fuzzified statuses to actuators'. Basically, the implementation of a fuzzy controller consists of the following steps:

1. we seek the help of the expert to choose the important variables for the controlling task; we suppose we have sensors able to measure each of these

variables; nonetheless, we ask the expert to "fuzzify" these inputs, i.e., to introduce fuzzy sets and membership functions; the same must be done for actuators;

2. a set of fuzzy *if-then* rules has to be written, so that every possible combinations of the input variables leads to an output in terms of actuators' statuses; if boolean or other logical combinations of the variables appear in the rules, a semantics to deal with these expressions must be given; also, a way to determine the value of the *then* part from the value of the *if* part must be given; it must be noted that this procedure produces many different outputs, one for every rule;

3. the outputs given by the rules must be combined to produce a single output (various methods for doing this have been proposed); this process is called "defuzzification".

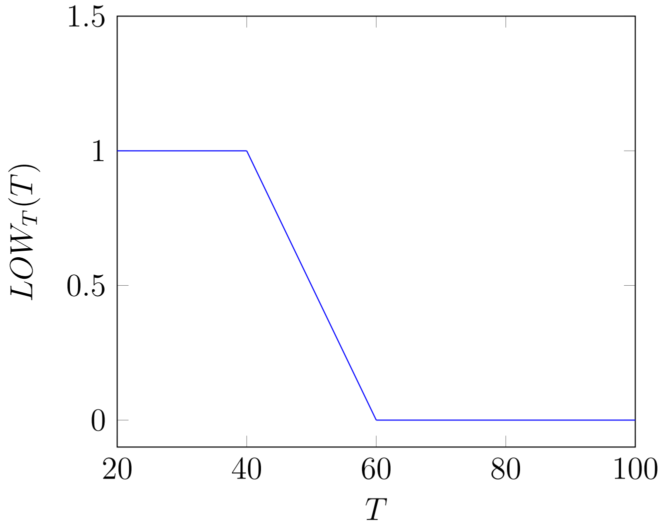
Let's explain better this procedure using an example. Suppose we have a system consisting of some fluid in a container, whose status is fixed by two parameters: temperature and pressure. Both can be measured by sensors, and both can be modified by actuators: a heater and a compressor. Suppose that temperature can vary between 20 degrees (when the heater is off) and 100 degrees (when the heater is functioning at maximum power), while pressure can vary between 1 bar (when the compressor is off) and 100 bars (when the compressor is at its maximum). Finally, suppose our goal is to keep the fluid as close as possible to a temperature of 50 degrees and a pressure of 50 bars. A fuzzy controller for this system could be implemented this way:

1. fuzzification; a possible choice for the fuzzy sets could be the following:

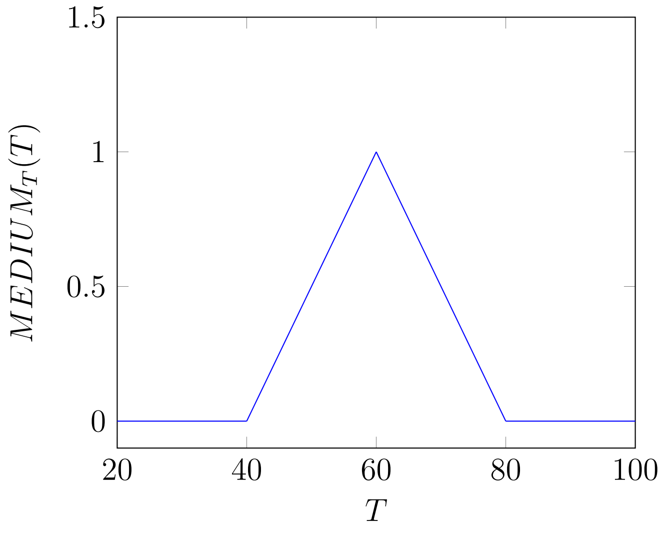
- measured temperatures in interval $[20, 46]$ and measured pressures in interval $[1, 33]$ are labeled as LOW;
- measured temperatures in interval $[47, 73]$ and measured pressures in interval $[34, 67]$ are labeled as MEDIUM;
- measured temperatures in interval $[74, 100]$ and measured pressures in interval $[68, 100]$ are labeled as HIGH;

analogous labels can be set for actuators functioning in corresponding intervals; the attribution of the values of the sensors and the actuators to the fuzzy sets can be done using these functions:

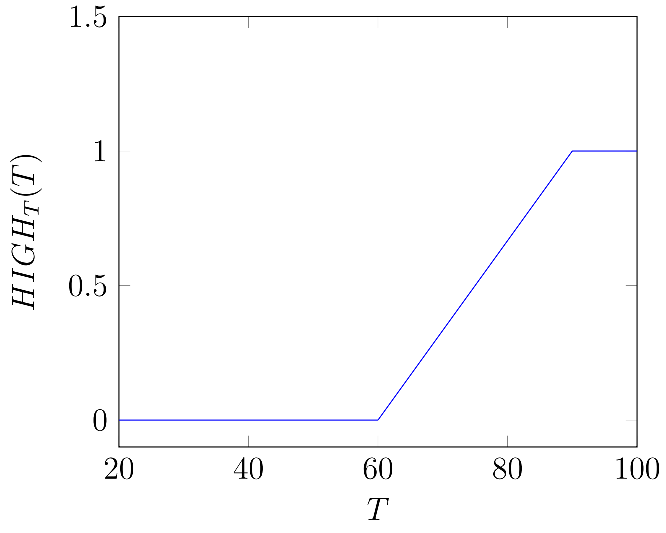
$$LOW_T(T) = \begin{cases} 1 \; for \; 20 \leq T \leq 40 \\ -\frac{T}{20} + 3 \; for \; 40 \leq T \leq 60 \\ 0 \; for \; 60 \leq T \leq 100 \end{cases} \tag{1.7}$$



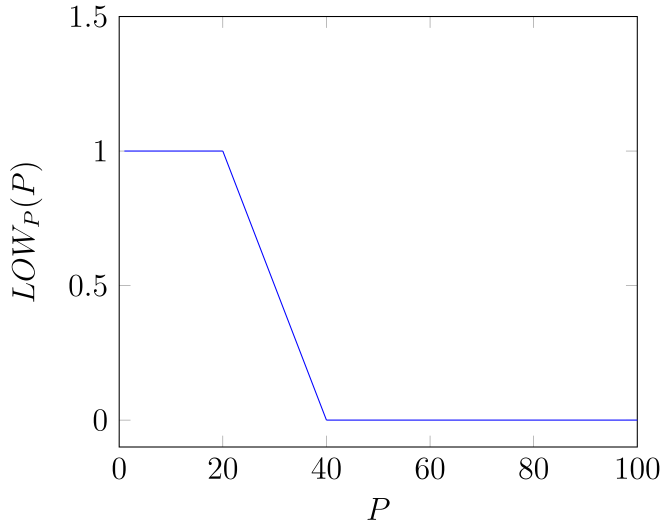
$$MEDIUM_T(T) = \begin{cases} 0 \; for \; 20 \leq T \leq 40 \\ \frac{T}{20} - 2 \; for \; 40 \leq T \leq 60 \\ -\frac{T}{20} + 4 \; for \; 60 \leq T \leq 80 \\ 0 \; for \; 80 \leq T \leq 100 \end{cases} \tag{1.8}$$



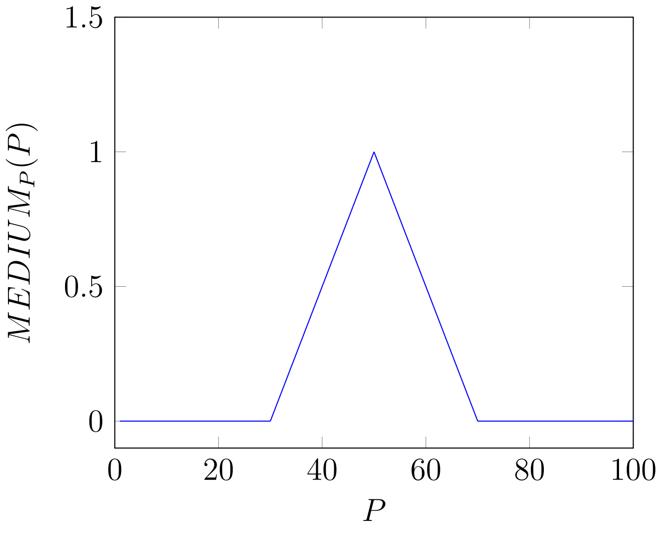
$$HIGH_T(T) = \begin{cases} 0 \; for \; 20 \leq T \leq 60 \\ \frac{T}{30} - 2 \; for \; 60 \leq T \leq 90 \\ 1 \; for \; 90 \leq T \leq 100 \end{cases} \tag{1.9}$$



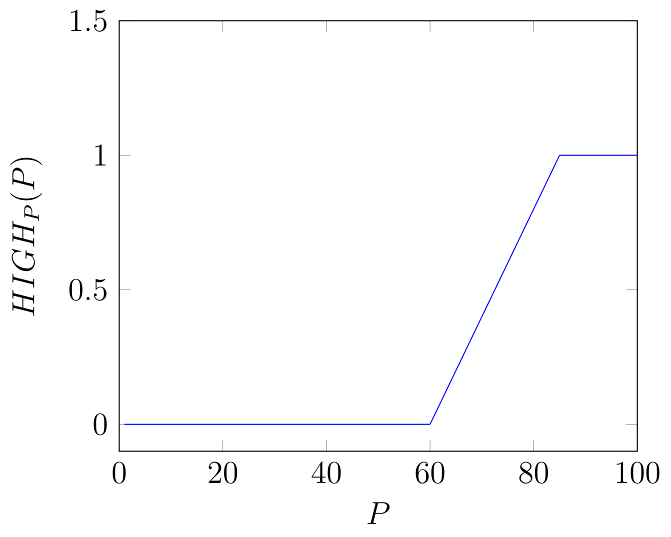
$$LOW_P(P) = \begin{cases} 1 \; for \; 1 \leq P \leq 20 \\ -\frac{P}{20} + 2 \; for \; 20 \leq P \leq 40 \\ 0 \; for \; 40 \leq P \leq 100 \end{cases} \tag{1.10}$$



$$MEDIUM_P(P) = \begin{cases} 0 \ for \ 1 \leq P \leq 30 \\ \frac{P}{20} - \frac{3}{2} \ for \ 30 \leq T \leq 50 \\ -\frac{P}{20} + \frac{7}{2} \ for \ 50 \leq P \leq 70 \\ 0 \ for \ 70 \leq P \leq 100 \end{cases} \tag{1.11}$$



$$HIGH_P(P) = \begin{cases} 0 \ for \ 1 \leq P \leq 60 \\ \frac{P}{25} - \frac{12}{5} \ for \ 60 \leq P \leq 85 \\ 1 \ for \ 85 \leq P \leq 100 \end{cases} \tag{1.12}$$



2. rules; we can implement the following rules (for economy, we indicate temperature with T, pressure with P, the heater with H and the compressor with C):

1)	T is LOW and P is LOW	H is HIGH and C is HIGH
2)	T is LOW and P is MEDIUM	H is HIGH and C is MEDIUM
3)	T is LOW and P is HIGH	H is MEDIUM and C is LOW
4)	T is MEDIUM and P is LOW	H is MEDIUM and C is MEDIUM
5)	T is MEDIUM and P is MEDIUM	H is MEDIUM and C is MEDIUM
6)	T is MEDIUM and P is HIGH	H is LOW and C is LOW
7)	T is HIGH and P is LOW	H is LOW and C is MEDIUM
8)	T is HIGH and P is MEDIUM	H is LOW and C is MEDIUM
9)	T is HIGH and P is HIGH	H is LOW and C is LOW

We use Zadeh's fuzzified version of boolean operators to find the truth value of the *if* part of every rule, and simply set the value of the *then* part to be equal to the value of the *if* part (the *then* part of every rule consists of two statements, one for the heater and one for the compressor; we set the value of each of them equal to the value of the *if* part of the rule);

3. defuzzification; we only consider the rule leading to the highest contribution.

This is all we need to make the controller work. If, for example, sensors read a temperature of 55 degrees and a pressure of 77 bars, then we have the following values for the membership functions:

LOW_T	0.25
$MEDIUM_T$	0.75
$HIGH_T$	0
LOW_P	0
$MEDIUM_P$	0
$HIGH_P$	0.68

From Zadeh's semantics we see that

$LOW_T \wedge LOW_P$	0
$LOW_T \wedge MEDIUM_P$	0
$LOW_T \wedge HIGH_P$	0.25
$MEDIUM_T \wedge LOW_P$	0
$MEDIUM_T \wedge MEDIUM_P$	0
$MEDIUM_T \wedge HIGH_P$	0.68
$HIGH_T \wedge LOW_P$	0
$HIGH_T \wedge MEDIUM_P$	0
$HIGH_T \wedge HIGH_P$	0

According to our rules, then, we have the following values for each possible output:

heater is HIGH and compressor is HIGH	0
heater is HIGH and compressor is MEDIUM	0
heater is MEDIUM and compressor is LOW	0.25
heater is MEDIUM and compressor is MEDIUM	0
heater is MEDIUM and compressor is MEDIUM	0
heater is LOW and compressor is LOW	0.68
heater is LOW and compressor is MEDIUM	0
heater is LOW and compressor is MEDIUM	0
heater is LOW and compressor is LOW	0

According to our defuzzification method, we only have to consider the outputs having the highest value, i.e., heater is LOW with membership value of *0.68* and compressor is LOW with membership value of *0.68*. Inverting functions LOW_T and LOW_P we see that

$$LOW_T(46.4) = 0.68, \tag{1.13}$$

$$LOW_P(26.4) = 0.68. \tag{1.14}$$

Our controller, then, prescribes to set the heater on 46.4 degrees and the compressor on 26.4 bars.[3] Few considerations on this example will allow us to better understand the main feature of fuzzy control. Our aim was to control temperature and pressure of a fluid in a container of given volume. As known, volume, temperature and pressure of a fluid are connected by means of the so called equation of state. If such equation were given for the considered fluid, traditional controllers would have used it to predict how temperature changes would affect pressure. This information would then be used to try to evaluate the best settings for the actuators. For this reason, these controllers are often said model-based: they predict changes in the system using a model. Fuzzy controllers use a different approach.

[3]Additional rules are needed for the intervals where the membership functions are not invertible; since this case does not show up in our example, we avoided to state them.

They do not suppose any model for the interaction between system's variables and between system and actuators. They use empirical data (what above has been said to be the expert's advising) to establish which option among the many possible is the better in every occasion. The model, in this way, is implicitly contained in the controller itself, namely, in the fuzzy rules. For example, in our treatment of the problem of controlling the fluid's temperature and pressure the connection between these two variables is acknowledged in rule 3, where, despite the necessity of increasing temperature using the heater, it is prescribed to keep this actuator at a medium level, in order to avoid an excessive increase of the already high pressure level. For this reason, fuzzy controllers are sometimes said to be model-free. This characteristic makes them usually simpler than their competitors; nonetheless, their performances are sometimes even better. That is the reason of the interest raised by them in the last years. Nonetheless, fuzzy controllers usually present some limitations, which we are going to discuss.

1.2.1 Present fuzzy controllers' limitations and our proposal

As anticipated in the previous section, one of the main differences between traditional controllers and fuzzy controllers is that traditional controllers are model-based (models for the system, and possibly for the environment and for the interaction between them are given), while fuzzy controllers are usually model-free (no attempt is made to try to describe the system's behavior, but rules of empirical validity are given). This difference is responsible for some of the positive aspects of fuzzy controllers. With respect to traditional controllers, fuzzy controllers are usually implemented more easily (they use few simple *if-then* rules instead of usually complex differential equations), require less memory, only taking into account what is really needed, and less calculations (differential equations often can be solved only numerically, and the algorithms which perform this task usually converge very slowly; in comparison, the fuzzification and defuzzification processes require very few calculations). But at the same time this feature has a downside: lacking a model of the interaction between the controller and the environment, for example, fuzzy controllers are unable to anticipate what can be the possible changes in the environment, and are bound to intervene only when the values measured by the sensors are outside the prescribed intervals. Let's clarify this statement with the example of a controller for the temperature inside a room. A model-free controller for this system could be a thermostat, which only measures the temperature inside the room and intervenes when it finds it outside a prescribed interval. Of course, such a controller cannot keep the temperature in the given interval all the time, because it is unable to foresee changes due to the variations of the temperature of the environment outside the room. A model-based controller would have sensors to measure the temperature of the environment, too, and a model which enables it to predict how the temperature outside influences that inside the room. Such a controller could intervene with its actuators before the changes of the environment would show their effects, and succeed in keeping the inside temperature always in the prescribed interval. Another limitation of fuzzy controllers as previously described is that they can only fulfill specifications of a static kind (a certain condition has to hold in the present instant, or as soon as possible), since the mechanism of the rules previously described does not allow to apply long-term strategy, which would make possible to handle more elaborate specifications. Still referring to the previous example, a specification such as *The temperature of the room never lies outside the prescribed interval* *for more than ten minutes*, could not be fulfilled by usual fuzzy controllers, since they do not take into account time. The main idea, which we pursue in the next chapters, is to define a model-based fuzzy controller, which retains the positive aspects of an ordinary fuzzy controller, to be simple and fast (i.e., to keep using the mechanism based on fuzzification, fuzzy rules and defuzzification), but implements some characteristics of the model-based controllers to be able to attend to a wider range of tasks, such as foreseeing changes in the environment and handling temporal specifications adopting long-term strategies. We plan to model the temporal aspects of the controlling task using Linear Temporal Logic (LTL), and to describe the interaction between the controller and the environment as a game. Chapters 2 and 3 will then deal with these subjects.

Chapter 2

Reactive systems and Linear-time Temporal Logic (LTL)

In this chapter we define reactive systems, introducing a model for their description, and introduce the basic concepts of temporal logic, with particular attention to its applications in specifying system properties.

2.1 Reactive systems

Systems are said to be “reactive” if they interact with their environment constantly. They are able to accept inputs and produce outputs all along their time of functioning, which is supposed to be infinite. To deal with reactive systems models are needed, and many have been proposed. The one we will use in this work, as described in the following subsections, is called *Kripke structure*.

2.1.1 Formal definition of Kripke structures

A Kripke structure $K = (S, E, I, f, s_o)$ is an oriented graph with no isolated vertices. In this graph:

- a function $f : S \to 2^I$ is given, where S is the set of *states* of the Kripke structure, corresponding to the possible states the system can visit, and I is a set of sentences, called *atomic propositions*; therefore, f label each state with a mark for every atomic proposition, indicating whether it is true or false in that state;
- oriented *edges* describe allowed transitions from one state to another; we indicate their set with letter E; we notice that $E \subseteq S \times S$;
- s_o is the *initial state*;
- every transition takes one instant of time to be performed.

A path π on a Kripke structure is an infinite sequence of states of the structure, $s_0 s_1 ... s_k ...$, such that, for every *n*, $(s_n,\ s_{n+1}) \in E$. The first state, s_0, is specified in K. The suffix π_i of path π is path $s_i ... s_k ...$. If p_i is the i-th atomic proposition and s_j a state of a Kripke structure, we use the symbol $s_j(p_i)$ to indicate the value of atomic proposition p_i in s_j, i.e., we set $s_j(p_i) = f_i(s_j)$, where f_i is the i-th component of the labeling function f. Kripke structures are very simple, yet they are general enough to describe a wide range of systems. Indeed, they are the models essentially used in control theory, being able to suitably describe computer networks, flexible manufacturing systems and start-up and shut-down procedures of industrial plants, just to make few examples (see [8]).

2.1.2 Usefulness of Kripke structures in modeling reactive systems

We are familiar with the concept of a system as a continuous medium, whose evolution over time is described by functions. Many systems, however, can be discretized. Suppose we are studying a system which, with good approximation, can be found in a finite set of different states, each characterized by the presence or absence of a given attribute, and that the system is able to go from any state to some others in a given, fixed amount of time. In this case, we could completely neglect what happens in the middle of transitions from one state to another, and only consider the ordained sequence of states visited by the system. Systems like these, therefore, can be suitably described by Kripke structures. Let's see how this can be done with an example.

2.1.3 Example of a reactive system described by a Kripke structure

Suppose we have a very simple microwave oven, whose state is completely defined by the value of two attributes: *ON/OFF* and *OPEN/CLOSED*. Suppose the user can open the oven when it is closed and off, close it when it is open, turn it on when it is closed and off and turn it off when it is closed and on. All these dynamics are then condensed in the following Kripke structure:

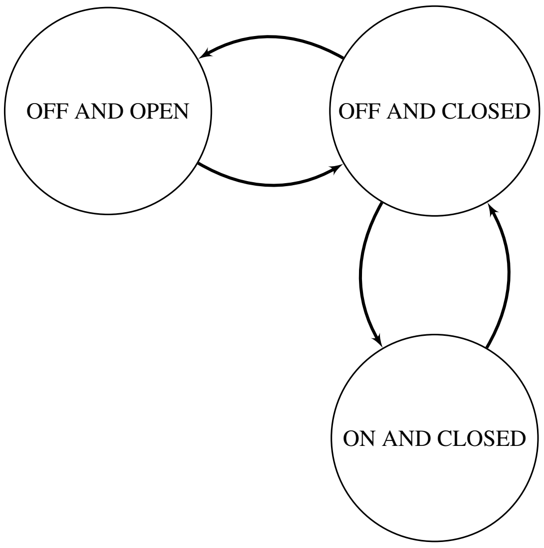


Figure 2.1: Kripke structure describing states and transitions of a microwave oven.

We see that the description provided by the Kripke structure is very synthetic, since it only takes into account the characteristics of the system which are really relevant.

2.2 Linear-time Temporal Logic (LTL)

In the preceding section we have seen how to usefully describe reactive systems. Now we want to introduce a way to state specifications for systems, i.e., to make statements on the possible paths of a given Kripke structure. We will do this using a particular kind of logic which allows us to talk about time.

2.2.1 Formal definition of LTL

LTL stands for “linear temporal logic” or “linear-time temporal logic”. Practically, LTL is built supposing a discrete concept of time is given (time is seen as a discrete, ordained succession of instants), and adding temporal operators to the usual boolean ones. Temporal operators usually implemented, the ones we will consider in the following, are these:

- Next ϕ ($\mathrm{X}\phi$): formula ϕ will hold in the next instant;
- Finally ϕ ($\mathrm{F}\phi$): ϕ holds at least one instant in the future;
- Always ϕ ($\mathrm{G}\phi$): from now on, ϕ will always hold;
- ϕ Until ψ ($\phi\mathrm{U}\psi$): ψ is true now or will be true in the future; ϕ has to be true at least until ψ becomes true.

We suppose given an amount of atomic propositions, for each of whom we suppose we know the truth value (for the moment we suppose propositions and formulas only can have the boolean truth values, i.e., *True* or *False*). The way a formula can be built in a given logic is called the *syntax* of the logic. A syntax can be conveniently described using a mathematical object known as *formal grammar*. We can write LTL syntax using the following, usual grammar:

$$\phi := True \,|\, False \,|\, p \,|\, \phi \wedge \phi \,|\, \neg\phi \,|X\phi \,|\, \phi \; U \; \phi, \tag{2.1}$$

where p is an atomic proposition. We have considered neither operator $\vee$, since it can be obtained from $\wedge$ and $\neg$ ($\phi \vee \psi = \neg(\neg\phi \wedge \neg\psi)$), nor operators F and G, since they can be obtained from U and $\neg$ ($\mathrm{F}\phi = \mathrm{True\ U}\ \phi$ and $\mathrm{G}\phi = \neg\mathrm{F}\neg\phi$). The truth value of a given formula has to be evaluated along the infinite succession of time instants, assuming we know the truth value of the atomic propositions at each instant. If we have described the system using a Kripke structure, then its temporal evolution corresponds to a path along the Kripke structure, each time step corresponding to a transition. The truth value of an atomic proposition at a given instant, then, is the truth value of that atomic proposition in the state visited by the system in that instant. We have already introduced a notation to indicate the value of an atomic proposition in a particular state of a Kripke structure in subsection 2.1.1. We now introduce the notation $[\![\pi, \phi]\!]$ to indicate the value of formula ϕ along path π. The way a meaning is given to any formula allowed by the syntax of a given logic is called the *semantics* of that logic. Let's see what LTL semantics is. We set the constants *True* and *False* to have values *1* and *0*, respectively, on every path. In our notation, we can state this fact with the following expressions:

$$[\![\pi, True]\!] = 1, \tag{2.2}$$

$$[\![\pi, False]\!] = 0. \tag{2.3}$$

Then, we set the truth value of an atomic proposition along a path to be equal to the truth value of that atomic proposition in the initial state of that path:

$$[\![\pi, p]\!] = s_0(p). \tag{2.4}$$

Suppose we know the value of formulas ϕ and ψ along π. Then we set the value of their conjunction $\phi \wedge \psi$ along π to be equal to 1 if both ϕ and ψ have value 1 along π, 0 otherwise. In other words we set:

$$[\![\pi, \phi \wedge \psi]\!] = [\![\pi, \phi]\!] \ \wedge \ [\![\pi, \psi]\!]. \tag{2.5}$$

Known the value of formula ϕ along π, $[\![\pi, \phi]\!]$, we set $[\![\pi, \neg\phi]\!]$ to be equal to 1 if $[\![\pi, \phi]\!] = 0$, equal to 0 otherwise, so that we can write:

$$[\![\pi, \neg\phi]\!] = \neg\ [\![\pi, \phi]\!]. \tag{2.6}$$

Provided we know how to evaluate a formula ϕ along any path, we set the value of $X\phi$ along a path π to be equal to the value of ϕ along path π_1:

$$[\![\pi, X\phi]\!] = [\![\pi_1, \phi]\!]. \tag{2.7}$$

Finally, if we recall the definition of Until given above, we see that we have to set $[\![\pi, \phi\ U\ \psi]\!] = 1$ if $[\![\pi, \psi]\!] = 1$, $\vee$ if $[\![\pi, \phi]\!] = 1 \wedge [\![\pi_1, \psi]\!] = 1$, and so on, while we will set $[\![\pi, \phi\ U\ \psi]\!] = 0$ if none of these eventualities holds. What said can be expressed by this expression:

$$[\![\pi, \phi\ U\ \psi]\!] = \vee_{i \geq 0}\{[\![\pi_i, \psi]\!]\ \wedge\ \wedge_{0 \leq j \leq i}[\![\pi_j, \phi]\!]\}. \tag{2.8}$$

These rules completely fix LTL semantics.

2.2.2 Usefulness of LTL in modeling specifications of reactive systems

LTL is very useful in stating temporal specifications for systems. Indeed, Amir Pnueli first introduced it in [9] to deal with formal verification of computer programs (for this contribution, Pnueli won a Turing Award). Let's just give an example. Suppose we have a computer program, and that we want the following statement to be true: *Any time the user executes the operation described by the the atomic proposition p="button A has been pushed", atomic proposition q="letter*

A has been written on the screen" becomes true in at most three instants of time. We can simply state this sentence using the LTL formula $G(p \to (Xq \vee XXq \vee XXXq))$. LTL allows to easily state some of the most common kinds of specifications for systems, such as:

- $safety$: an undesired eventuality never happens, $G(\neg\phi)$;
- $liveness$: a desired eventuality keeps happening in the future, $G(F\phi)$.

Both these properties can be checked as particular cases of the so-called model checking problem: given an LTL formula and a Kripke structure, verify that every path on that structure satisfies that formula. In the past years efficient ways to solve the model checking problem for LTL have been proposed.

2.2.3 Fuzzification of LTL

In Subsection 1.1.1 we have already seen how to extend ordinary boolean operators in order to deal with fuzzy values. Having given LTL semantics in terms of boolean operators, the same can be done for LTL: we only have to substitute boolean operators with their fuzzy version. Therefore, we see that a fuzzy version of LTL compatible with Zadeh's fuzzy boolean operators is defined by the following semantics:

$$[\![\pi, \neg\psi]\!] = 1 - [\![\pi, \psi]\!], \tag{2.9}$$

$$[\![\pi, \psi_1 \wedge \psi_2]\!] = \max\left\{[\![\pi, \psi_1]\!], [\![\pi, \psi_2]\!]\right\}, \tag{2.10}$$

$$[\![\pi, \psi_1\ U\ \psi_2]\!] = \sup_{i \geq 0}\left\{\min\left\{[\![\pi_{\mathrm{i}}, \psi_2]\!], \min_{0 \leq j < i}[\![\pi_{\mathrm{j}}, \psi_1]\!]\right\}\right\}, \tag{2.11}$$

(constants *True* and *False* and atomic propositions are evaluated in the same way as in boolean LTL; operator X, also, has the same semantics). We refer to this fuzzified version of LTL as to LTL[Z], the Z staying for “Zadeh”.

Chapter 3

Fuzzy LTL games

This chapter starts with an exposition of the basic concepts of automata theory. A presentation of the traditional automata-based technique for solving the model-checking problem for LTL specifications follows. In Section 3.3 LTL games are defined, and the usual way to solve them via automata is exposed. Finally, in Section 3.4, we introduce a variation of these kinds of games, which we call fuzzy LTL games, and show how the technique used for LTL games can be adapted to solve these ones. It provides the first original contribution of this work.

3.1 Basic notions of automata theory

Automata are mathematical objects capable of representing computational problems in a very simple but effective way. For this reason, they are a fundamental tool in computer science. In this section we will give the basic notions of automata theory, limiting ourselves to the aspects of the theory needed for our purposes.

3.1.1 Formal definitions

In order to define automata and their properties, we recall the following definitions from formal languages' theory:

- an *alphabet* Σ is a collection, finite or infinite, of *symbols*;
- a *word* ω over an alphabet Σ is a sequence, finite or infinite, of elements of Σ;
- a language L over an alphabet Σ is a collection of words over Σ.

An automaton is a tuple $\langle \mathrm{Q}, \Sigma, \delta, \mathrm{q}_0, \mathrm{F} \rangle$, where:

- Q is a set of states; we denote the generic element of Q by q;
- Σ is a finite alphabet; we denote the generic element of Σ by τ;
- $\delta : \mathrm{Q} \times \Sigma \to 2^{\mathrm{Q}} \setminus \varnothing$ is a transition rule: given a state q and a symbol τ, $\delta(\mathrm{q}, \tau)$ is the set of possible next states; note that δ can be single-valued or many-valued: in the former case the automaton is said to be *deterministic*, in the latter it is said to be *non-deterministic*;
- $\mathrm{q}_0 \in \mathrm{Q}$ is the initial state;
- $\mathrm{F} \subseteq \mathrm{Q}$ is the set of the *accepting states*.

Automata take words $\tau_0\tau_1...\tau_{\mathrm{n}}...$ as inputs, and produce *runs* $q_0q_1...q_n...$, i.e. sequences of states starting from the initial state such that, for every *n*, $\mathrm{q}_{\mathrm{n}+1} \in \delta(\mathrm{q}_{\mathrm{n}}, \tau_{\mathrm{n}})$. A word, submitted to a deterministic automaton, produces one run. In the case of non-deterministic automata, instead, many runs can correspond to the same input word. An automaton is said *finite* if it has a finite number of states, *infinite* otherwise. Both finite and infinite words can be submitted to an automaton. Sometimes, the word “automaton” is used to indicate those which work on finite words, while those using infinite words as input are called *ω-automata*. In the following, we will only consider finite ω-automata. A finite non-deterministic ω-automaton is usually indicated with $\mathrm{NF}\omega$, while a finite deterministic ω-automaton is usually referred to as $\mathrm{DF}\omega$. Among all the possible runs of an automaton some are said to be *accepting*. A word is said to be *accepted* by the automaton if there is an accepting run on it. The rule which prescribes which runs are accepting is called the *accepting condition* of the automaton. Many different accepting conditions have been considered, leading to many different types of automata. For the moment we only consider two of them. Firstly, said ρ a run, we define $Inf(\rho)$ as that subset of Q consisting of states which appear an infinite number of times in ρ. The two accepting conditions can then be stated as follows:

Büchi accepting condition. *A run ρ is accepted if at least one accepting state appears in $Inf(\rho)$.*

Parity accepting condition. *Let the states of the automaton be labeled with natural numbers from 0 to m; a run ρ is accepted if the smallest label number appearing in $Inf(\rho)$ is even.*

A non-deterministic Büchi ω-automaton is usually indicated as $\mathrm{NB}\omega$, a deterministic one with $\mathrm{DB}\omega$. Analogously, $\mathrm{NP}\omega$ and $\mathrm{DP}\omega$ indicate respectively non-deterministic and deterministic parity ω-automata. A word ω is said to be accepted by an automaton A if a run ρ produced by A when processing the input ω is accepted. The set of words over an alphabet accepted by an automaton forms, according to the definition given above, a language. This language is said to be *recognized* by the automaton.

3.1.2 Automata problems of interest in computer science

Typical problems in automata theory consist in establishing properties of the languages of certain automata. In the following we will deal with the following two:

- *emptiness*: deciding if a certain automaton accepts any word at all, i.e., if its language is the empty set or not;
- *inclusion*: deciding whether the language of one automaton is included into the language of another or not.

We also introduce a slightly different kind of problem which will play a role in the following:

- *determinization*: given a non-deterministic automaton, construct a deterministic one which recognizes the same language.

Typically determinization, when possible, implies an exponential blow-up in the number of states and transitions of the automaton. In some cases, it also impose to change the accepting condition. Indeed, it is known that non-deterministic Büchi automata are more expressive than deterministic Büchi automata. It means that it is impossible in general to find a deterministic Büchi automaton which recognizes the same language of a non-deterministic Büchi automaton. But it has been proven that deterministic parity automata are as expressive as non-deterministic parity automata, which in turns are more expressive than non-deterministic Büchi automata. It means that the following theorem holds:

Theorem 1 (from NBω to DPω). *For every NBω $\mathrm{A} = \langle \mathrm{Q}, \Sigma, \delta, \mathrm{q_o}, \mathrm{F} \rangle$ exists a DPω $\widetilde{\mathrm{A}} = \left\langle \widetilde{\mathrm{Q}}, \Sigma, \widetilde{\delta}, \widetilde{\mathrm{q}_0}, \mathrm{P} \right\rangle$ such that A and $\widetilde{\mathrm{A}}$ recognize the same language (P is the function assigning priorities to every state in $\widetilde{\mathrm{Q}}$).*

For the construction of $\widetilde{\mathrm{A}}$ see, for example, [10].

3.2 Model-checking problem for LTL specifications

In this section we give a precise definition of the model-checking problem for LTL specifications, and show how it can be solved applying the Vardi-Wolper procedure. As we will see, this procedure permits to construct an automaton which accepts only computations which satisfy a given formula.

3.2.1 Formal statement of the model-checking problem

The model-checking problem can be stated as follows:

Statement of the model checking problem. *Given a Kripke structure $K = (S, E, I, f, s_0)$ and a temporal logic formula ϕ constructed on the set I of atomic propositions of K, decide whether or not, for every path π in K starting from s_0, $[\![\pi, \phi]\!] = 1$.*

Some model-checking algorithms, when the answer to the model-checking problem is negative, produce a counterexample.

3.2.2 Solving the model-checking problem using the Vardi-Wolper procedure

In their 1983 paper [11] Vardi and Wolper showed, given an LTL formula ϕ, how to construct an automaton A_ϕ which only accepts computations satisfying ϕ. We do not go into the details of the construction, nor in the proof of its correctness, but just show how it can be used to solve the model-checking problem:

1. following Vardi and Wolper, building a non-deterministic Büchi automaton A_ϕ which only accepts computations satisfying ϕ;

2. building a non-deterministic Büchi automaton A_K which accepts all and only computations of K;

3. checking whether the language of $\mathrm{A_K}$ is included in the language of A_ϕ; clearly, this would assure that every computation in K starting from $\mathrm{s_0}$ satisfies ϕ.

3.3 LTL games

We are going to define LTL games, and to show how the objects used to solve the model-checking problem can be applied to solve them.

3.3.1 Formal definitions of LTL, Büchi and parity games

In this section we define three different kinds of games, not only LTL. The reason is that we will solve LTL games reducing them to other games, namely Büchi or parity. These three games share many features, namely:

- they are played on a Kripke structure $\mathrm{K} = (\mathrm{S}, \mathrm{E}, \mathrm{I}, \mathrm{f}, \mathrm{s_0})$, called the *arena*;
- they are played by two players; the set of vertices S is partitioned into two subsets, $\mathrm{S_0}$ and $\mathrm{S_1}$; player 0 moves when in a state belonging to $\mathrm{S_0}$, choosing one of the possible transitions from that state; similarly, player 1 moves when in a state belonging to $\mathrm{S_1}$;
- player 0 wins when the infinite path π in K generated by the players' moves satisfies some condition, called the *winning condition*; player 1 wins if player 0 fails.

In Büchi and parity games, we suppose the states labeled as said in Subsection 3.1.1. The only difference between LTL, Büchi and parity games is in the winning conditions:

- in LTL games, an LTL formula ϕ in terms of the atomic propositions in I is given; the winning condition for player 0 is: $[\![\pi, \phi]\!] = 1$;
- in Büchi games, the winning condition for player 0 is: at least one accepting state appears infinitely often in π;
- in parity games, the winning condition for player 0 is: among the states which appear infinitely often in π, the one with the highest priority has an even priority.

It is useful to introduce the concept of *strategy*, which we define as follows:

Definition of strategies. *Be* Π *the set consisting of all the possible finite paths of the form* $s_j...s_k$ *on a Kripke structure* $K(S, E, I, f, s_0)$. *A strategy for a player* i *is a function* $\sigma : \Pi \to S$ *compatible with the transitions in* K*, i.e., such that for every* $n \in \mathbb{N}$ *we have* $\sigma(s_j...s_n) = s_{n+1} \in \mathrm{S}$ *and* $(s_{n+1}, \sigma(s_j...s_n)) \in E$. *A strategy is said to be memoryless if* $\sigma(s_j...s_n) = \sigma(s_n)$ *for all* $s_j...s_n$. *A path* $\pi = s_j...s_n...$ *is said to be a play generated by* σ *if, for every n such that* s_n *is a state belonging to player i,* $s_{n+1} = \sigma(s_j...s_n)$. *A strategy for player* i *is said to be winning if every play generated by it is winning.*

We define player 0's winning region as the subset P_0 of the set of states of the arena from which player 0 has a winning strategy; player 1's winning region P_1 has a similar definition. We recall this fundamental property of Büchi and parity games (see for example [12] and [13]:

Theorem 2 (winning strategies for Büchi and parity games). *In Büchi and parity games* P_0 *and* P_1 *are computable and form a partition of the set of states of the game's arena; when a player has a winning strategy, she has a memoryless one from every state in her winning region.*

3.3.2 Solving LTL games

The usual way to solve LTL games (see, for example, [14]) consists in translating them into games with a different type of winning condition which is known how to solve. Informally, the idea at the basis of this translation is to form the product between the arena where the game is played and the automaton $\mathrm{A}_\phi = \langle \mathrm{Q}, \Sigma, \delta, \mathrm{q}_0, \mathrm{F} \rangle$ defined in Subsection 3.2.2. The product set so obtained is seen as the arena of another game. This game is defined so to assure that winning it we both respect the transitions on the arena and the accepting condition of the automaton, therefore winning the initial LTL game. We recall that the automaton A_ϕ is generally non-deterministic: it happens to be deterministic only for particular formulas ϕ. This fact represent a problem, because the game on the product set is not well-defined unless the automaton involved in the product is deterministic. When A_ϕ is non-deterministic, then, it is necessary to determinize it. A deterministic automaton A_ϕ leads to a Büchi game. When determinization is needed, the kind of game the LTL game is translated into depends on the determinization we perform. The determinization we choose is from Büchi to parity. Consequently, the LTL game will result translated into a parity game. As before, $\mathrm{K} = (\mathrm{S}, \mathrm{E}, \mathrm{I}, \mathrm{f}, \mathrm{s}_0)$ is the arena where the LTL game is played. We construct a parity game $\mathrm{G} = (\mathrm{X} = \mathrm{X}_0 + \mathrm{X}_1, \widetilde{\mathrm{s}_0}, \widetilde{\mathrm{P}})$ according to the following steps (for generality, we suppose A_ϕ is non-deterministic):

1. translate A_ϕ into a deterministic parity automaton $\widetilde{\mathrm{A}}_\phi = \left\langle \widetilde{\mathrm{Q}}, \Sigma, \widetilde{\delta}, \widetilde{\mathrm{q}_0}, \mathrm{P} \right\rangle$;
2. form the set $\mathrm{X} = \mathrm{S} \times \widetilde{\mathrm{Q}}$; we suppose X partitioned into sets X_0 and X_1 according to S's partitioning, i.e., X_0 is formed by states (s, q) such that s is in S_0 and X_1 is formed by states (s, q) such that s is in S_1;

3. introduce transitions on X in the following way: from a state (s, q) it is possible to go to a state $(\mathrm{s}', \mathrm{q}')$ if and only if there is a transition in K which leads from s to s' and $\widetilde{\delta}(\mathrm{q}, \mathrm{f}(\mathrm{s})) = \mathrm{q}'$;

4. $\widetilde{\mathrm{P}}$ assigns to state (s, q) the same priority which P assigns to state q;

5. $\widetilde{\mathrm{s}_0} = (\mathrm{s}_0, \mathrm{q}_0)$, where q_0 is the initial state of $\widetilde{\mathrm{A}}_\phi$, is the initial state of G.

G is a parity game. A play in G is obtained by a play in K, and viceversa a play in G fixes a play in K. A winning strategy in G is guaranteed to produce a path $(\mathrm{s}_0, \mathrm{q}_0)(\mathrm{s}_1, \mathrm{q}_1)...(\mathrm{s}_\mathrm{n}, \mathrm{q}_\mathrm{n})...$ in X such that $\mathrm{s}_0\mathrm{s}_1...\mathrm{s}_\mathrm{n}...$ is a path compatible with K's structure, and $\mathrm{q}_0\mathrm{q}_1...\mathrm{q}_\mathrm{n}...$ is an accepting run for $\widetilde{\mathrm{A}}_\phi$, i.e., a computation satisfying ϕ. Therefore, if $(\mathrm{s}_0, \mathrm{q}_0)(\mathrm{s}_1, \mathrm{q}_1)...(\mathrm{s}_\mathrm{n}, \mathrm{q}_\mathrm{n})...$ is a winning play for Player 0 in X, then $\mathrm{s}_0\mathrm{s}_1...\mathrm{s}_\mathrm{n}...$ is a winning play for Player 0 in the original LTL game.

3.4 Fuzzy LTL games

Now we are ready to introduce and solve fuzzy LTL games.

3.4.1 Formal definition of fuzzy LTL games

Be $\mathrm{K} = (\mathrm{S} = \mathrm{S}_0 + \mathrm{S}_1, \mathrm{E}, \mathrm{I}, \mathrm{f}, \mathrm{s}_0)$ a Kripke structure as the ones introduced above, with the difference that f takes values in the interval $[0, 1]$. Be ϕ an LTL[Z] formula, i.e., a formula whose value on a path $\mathrm{s}_0\mathrm{s}_1...\mathrm{s}_\mathrm{n}...$ in K is evaluated according to Zadeh's semantics. We give the following definition:

Definition of LTL[Z] games. *An LTL[Z] game* A *is an infinite two-player game on an arena* K*. Player 0 chooses which move to take in states belonging to* S_0*, while Player 1 choose which move to take when in a state belonging to* S_1*. Player*

0's goal is to assure that the path produced by the moves of the two players is such that $[\![\pi, \phi]\!]$ has the highest possible value. Player 1's goal is to assure that the path produced by the moves of the two players is such that $[\![\pi, \phi]\!]$ has the lowest possible value.

In LTL[Z] games we do not talk about winning positions. Instead, we define the *value* $val(A, \phi)$ of an LTL[Z] game whose specification is formula ϕ:

$$val(A, \phi) = \sup_{\sigma} \inf_{\pi} [\![\pi, \phi]\!], \tag{3.1}$$

where σ is a strategy for player 0 and π is a path generated following σ. Informally, this definition means that the value of the LTL[Z] game considered is equal to the highest value player 0 can attain for ϕ when starting from s_0, when player 1 plays in the best possible way. In the following subsection we will show how to adapt the technique introduced in Subsection 3.3.2 to solve LTL[Z] games.

3.4.2 Solving fuzzy LTL games

At the beginning of this chapter we recalled that fuzzy LTL games are a new topic, so the way to solve them cannot be found in the literature. Nonetheless, fuzzy versions of LTL have already been given in the past, and the generalized model-checking problem for them has been solved, for example in [15]. We are going to adapt the main results of [15] to LTL[Z], so to have a good starting point to solve LTL[Z] games. The first step is bounding the number of values that a given specification can have on the given Kripke structure. This is done in Lemma 2.5 of [15], proven for an entire family of quantitative extensions of LTL which includes LTL[Z]. Restricting ourselves to LTL[Z], the limit on the number of values that a formula can have is lower than the one given in [15] for the general case. We are going to prove this assertion adapting the proof in [15]:

Lemma 1 (**bounding the number of values for LTL[Z] formulas**). *Given an LTL[Z] formula ϕ and a Kripke structure K, the number of values of the form $[\![\pi, \phi]\!]$, where π is a path on K, is finite, and is equal to $x \cdot |\phi|$, where x is the highest number of different possible values an atomic proposition can assume on K, and $|\phi|$ is the length of formula ϕ.*

Proof. Let's proceed by induction. The only formulas of length 1 are *True*, *False* and the atomic propositions. By the definition of *x*, for these propositions it trivially holds that $\#V(\phi) \leq x$. A formula of length *n* can be obtained:

1. via negation of a formula of length $\mathrm{n} - 1$;
2. via conjunction of formulas ψ_1 and ψ_2 such that $|\psi_1| + |\psi_2| = \mathrm{n} - 1$;
3. applying operator X to a formula of length $\mathrm{n} - 1$;
4. applying operator U to formulas ψ_1 and ψ_2 such that $|\psi_1| + |\psi_2| = \mathrm{n} - 1$;

Let's discuss each of these cases:

1. according to our semantics, $[\![\pi, \neg\psi]\!] = 1 - [\![\pi, \psi]\!]$; the set $\mathrm{V}(\neg\psi)$ is then formed by all numbers of the form $1 - \mathrm{y}$, where $\mathrm{y} \in \mathrm{V}(\psi)$; $\mathrm{f(y)} = 1 - \mathrm{y}$ is a one-to-one correspondence between $\mathrm{V}(\psi)$ and $\mathrm{V}(\neg\psi)$, so we conclude that $\mathrm{V}(\psi)$ and $\mathrm{V}(\neg\psi)$ have the same cardinality; our lemma then holds, even with a stronger bound;
2. since $[\![\pi, \psi_1 \wedge \psi_2]\!] = \max\{[\![\pi, \psi_1]\!], [\![\pi, \psi_2]\!]\}$, we have $\mathrm{V}(\psi_1 \wedge \psi_2) \subseteq \mathrm{V}(\psi_1) \cup \mathrm{V}(\psi_2)$, and therefore $\#\mathrm{V}(\psi_1 \wedge \psi_2) \leq \#\mathrm{V}(\psi_1) + \#\mathrm{V}(\psi_2)$; by the induction hypothesis, then, $\#\mathrm{V}(\psi_1 \wedge \psi_2) \leq \mathrm{x} \cdot (|\psi_1| + |\psi_2|) = \mathrm{x} \cdot (\mathrm{n} - 1)$;

3. the set $\mathrm{V}(\phi)$ contains the values assumed by formula ϕ along any possible path; for every possible path π, π_1 is a possible path too[1]; thus, $[\![\pi_1, \phi]\!]$ belongs to $\mathrm{V}(\phi)$ whatever path π we consider; since numbers of the type $[\![\pi_1, \phi]\!]$ form the set $\mathrm{V}(\mathrm{X}\phi)$, we conclude that $\mathrm{V}(\mathrm{X}\phi) \subseteq \mathrm{V}(\phi)$, and again the lemma holds in an even stronger form;

4. we have $[\![\pi, \psi_1 \mathrm{U} \psi_2]\!] = \sup\limits_{\mathrm{i} \geq 0} \left\{ \min \left\{ [\![\pi_\mathrm{i}, \psi_2]\!], \min\limits_{0 \leq \mathrm{j} < \mathrm{i}} [\![\pi_\mathrm{j}, \psi_1]\!] \right\} \right\}$, so again $\mathrm{V}(\psi_1 \mathrm{U} \psi_2) \subseteq \mathrm{V}(\psi_1) \cup \mathrm{V}(\psi_2)$ and $\#\mathrm{V}(\psi_1 \mathrm{U} \psi_2) \leq \#\mathrm{V}(\psi_1) + \#\mathrm{V}(\psi_2) \leq \mathrm{x} \cdot (\mathrm{n} - 1)$.

We notice that for formulas of length greater than 1 we have the stronger bound $\#V(\phi) \leq x \cdot (|\phi| - 1)$. We had to use the weaker bound in the statement of the theorem so that it could apply to formulas of length 1 too. □

Another result we will need from [15] is this:

Theorem 3 (extending the Vardi-Wolper procedure to LTL[Z] formulas). *For every LTL[Z] formula ϕ and for every interval $P \subseteq [0, 1]$ exists a non-deterministic Büchi automaton A_ϕ which accepts all and only computations such that ϕ's value is in P.*

Again, they prove their result for an entire family of fuzzy LTLs, but their procedure is general and we can apply it without modifications to the case of LTL[Z]. For this reason, we give and accept the statement without repeating the proof. From Lemma 1 we see that given a Kripke structure K and an LTL[Z] formula ϕ, a finite set $\{\mathrm{V}_1, ..., \mathrm{V}_\mathrm{k}\}$ of possible values for ϕ in K is given. Applying Theorem 3 to the k intervals I_j, each reducing to the single value V_j ($\mathrm{j} \leq \mathrm{k}$), we obtain a finite set of automata $\{\mathrm{A}_\phi{}^1, ..., \mathrm{A}_\phi{}^k\}$ such that, for every j, $\mathrm{A}_\phi{}^j$ accepts all and

[1]Of course, π and π_1 have different initial states, but we are considering any path, not only paths starting in a designated state.

only those computations such that the value of ϕ is $\mathrm{V_j}$. Some of these automata's languages might be empty: it means that the corresponding numbers $\mathrm{V_j}$ are not actually obtainable. We eliminate from sets $\{\mathrm{V_1}, ..., \mathrm{V_k}\}$ and $\{\mathrm{A}_\phi{}^{1}, ..., \mathrm{A}_\phi{}^{k}\}$ those indexes for which this eventuality occurs. Let $\{\mathrm{W_1}, ..., \mathrm{W_m}\}$ and $\{\mathrm{A}_\phi{}^{1}, ..., \mathrm{A}_\phi{}^{m}\}$, with $m \leq k$, be the sets obtained after this elimination. The value of the LTL[Z] game can be found using the same procedure described in Subsection 3.3.2: the only difference is that we might be obliged to repeat it up to m times. Indeed, when dealing with LTL[Z] games, we can build m different parity games, one for every automaton $\mathrm{A}_\phi{}^{j}$. Said j the highest index between 1 and m such that the corresponding game can be won by player 0, we have that $\mathrm{val(K}, \phi) = \mathrm{W_j}$. The best strategy for the LTL[Z] game can be obtained by the parity game corresponding to automaton $\mathrm{A}_\phi{}^{j}$ as done in Subsection 3.3.2. The results of this last section, which constitutes the first theoretical contribution of this work, can be seen as the proof of the following theorem:

Theorem 4 (decidability of LTL[Z] games). *Solving LTL[Z] games is decidable.*

It should be possible to prove that the complexity of this problem is *2EXPTIME-complete*, the lower bound being obtainable, for example, from [16], and the upper bound coming from the description of our algorithm.

Chapter 4

LTL[Z] controller synthesis

Now we have all the items needed to introduce our idea, the core of this work: an LTL[Z]-based controller. In Section 4.1 we illustrate step by step how to construct it, while in Section 4.2 we give a simple example.

4.1 Construction of an LTL[Z]-based controller

We suppose given a system y, which we aim to control with actuators in order to fulfill some LTL[Z] formula ϕ, and the environment e it is immersed in. We can split the construction of an LTL[Z]-based controller in two parts: the modeling process and the strategy synthesis. The modeling process consists in the following steps:

1. individuate a set of attributes I which completely specifies the state of s and e; formula ϕ must be expressed in terms of these attributes, too;

2. discretize y and e so that we can assume that they can only be in a finite number of states, which we call Y and E respectively, according to the

values of the attributes assigned to them by a labeling function f; be these values expressed in a scale going from 0 to 1; be the time discretized as well, and be t the shortest amount of time we consider; in the cartesian product $Y \times E$ individuate the only couples (Y_i, E_j) formed by states which are simultaneously observable; be W the subset of $Y \times E$ so obtained;

3. construct a set V duplicating every state in W; the set V be partitioned in the subsets V_0 and V_1 such that just one copy of every state belong to each of them; these sets represent states belonging to the controller and states belonging to the environment respectively;

4. let's introduce transitions T in V; the rules to be followed in doing so are:

   (a) no state can remain isolated;

   (b) a state in V_0 must necessarily lead to a state in V_1, and viceversa;

   (c) transitions must represent variations in the attributes for y which is possible to attain using the actuators, and variations in the attributes for e which can realistically happen in an amount of time t.

Now we discuss strategy synthesis. Before describing the steps it requires, we remind that, from the algorithm described in Chapter 3, solving an LTL[Z] game on an arena V is equivalent to solve a parity game on the arena $V \times Q$, where Q is the set of states of a certain automaton, and that the parity game on this larger arena can be solved applying a memoryless strategy.

1. For every state y_0 in V, solve the LTL[Z] game on the arena $K = (V = V_0 + V_1, T, I, f, y_0)$ corresponding to formula ϕ, i.e., establish what is the highest possible value for ϕ on a path starting from y_0 and find a strategy to assure it is obtained;

2. following Sections 3.3 and 3.4 expand the set of states of the controller from V to $V \times Q$;
3. note that the strategy on $V \times Q$ obtained in step 1, being memoryless, is equivalent to a set of fuzzy $if - then$ rules on $V \times Q$;
4. set the actuators response to sensors' inputs according to the rules obtained in step 3.

4.2 An example

In this example we want to synthesize a controller for the temperature of a room in a building. The system y is therefore the room, while the environment e is everything around it. The only variable we want to control is the temperature T. We only consider three atomic propositions: $T\ is\ LOW$, $T\ is\ MEDIUM$ and $T\ is\ HIGH$. To assign a value to these atomic propositions we label each state with an attribute between LOW, $MEDIUM$ and $HIGH$, and use the following table:

T	LOW	$MEDIUM$	$HIGH$
LOW	1	0.5	0
$MEDIUM$	0.5	1	0.5
$HIGH$	0	0.5	1

It means that the atomic propositions $T\ is\ LOW$ has value 1 in states labeled with LOW, value 0.5 in states labeled with $MEDIUM$ and value 0 in states labeled with $HIGH$, and so on. Following the procedure described in the preceding section, we obtain a Kripke structure like this for the arena of our game:

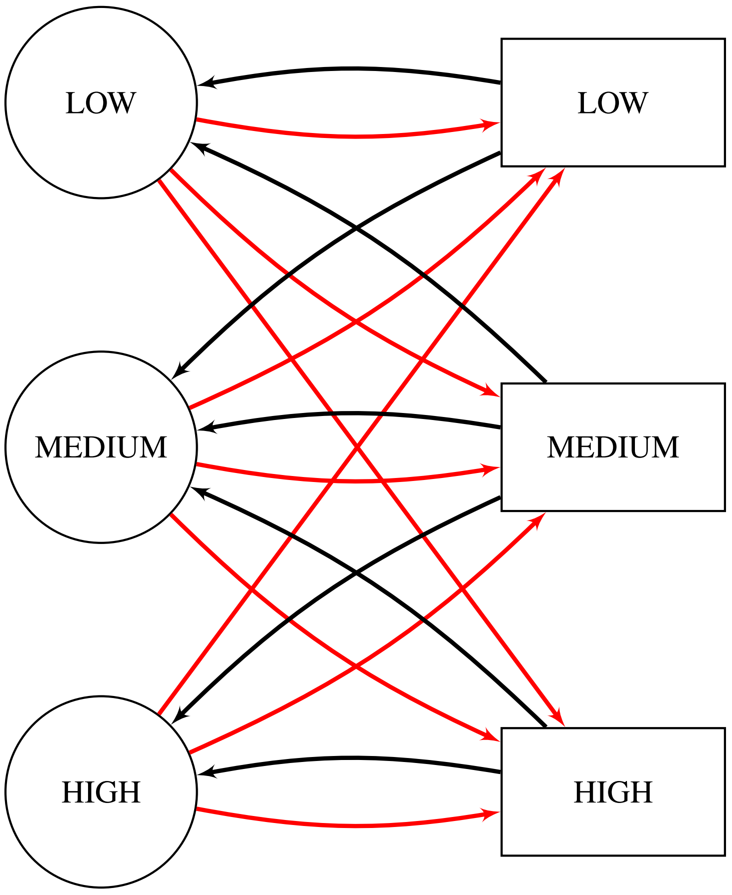


Figure 4.1: Kripke structure describing states and transitions for a system room-environment.

We have represented the states belonging to the controller with circles, and the states belonging to the environment with rectangles. It is easily verified that we have respected all the rules we have stated for the transitions, which, to make the figure more readable, we have colored in red when they start from a state belonging to the controller and in black when they start from a state belonging to the environment. To express the fact that we do not expect the environment to change too quickly we have forbidden it to go from LOW to $HIGH$ and viceversa, while we have given this possibility to the controller. A possible specification for the controller could be: $''G((T\ is\ LOW) \vee (T\ is\ HIGH)) \rightarrow X(T\ is\ MEDIUM)''$. We could choose a starting state, for example LOW, belonging to the environment, find the value of the corresponding game and use the technique exposed in Subsection 3.4.2 to find a winning strategy. According to the procedure described in the previous section, this strategy would then tell us which moves to take at every instant of time.

Chapter 5

Fragments of LTL

In Chapter 3 we have seen that solving an LTL game, or an LTL[Z] game, requires, in the general case, the determinization of a Büchi automaton. This step may present great difficulties, as briefly explained in Section 5.1. In the case of LTL, it has been found that determinization can be avoided when certain limitations are imposed on the form of the LTL formula the game is played with. These results are reported in Section 5.2. In Section 5.3 we show that something similar can be said for LTL[Z] games. Adapting the arguments valid for LTL to LTL[Z], we prove indeed that determinization can be avoided in some important cases with no much greater computational cost with respect to LTL. This proof constitutes the second original theoretical contribution of this work.

5.1 Difficulties in the determinization of a Büchi automaton

Determinization of non-deterministic automata is a very complex task. As reported, for example, in [17], only in 1988 Safra introduced an algorithm to determinize Büchi automata. His construction is found to be difficult, and leads to an exponential blow-up of the number of states. Moreover, as we said in Chapter 3, it inevitably requires to switch from Büchi to different accepting conditions, like parity. Lately, other constructions have been given, such as the one described in [10], which we have referred to in Chapter 3. We preferred it over the others since it led from LTL games to parity games, for which many efficient algorithms are known, but it still requires a huge increase in the number of states, which make it untractable for large systems. As pointed out in [14], the complexity of the determinization depends on the form of the LTL formula. For some less complex formulas, they notice that the Büchi automaton produced applying the Vardi-Wolper procedure is indeed deterministic. In this case determinization is unneeded, and the procedure to solve the game is much simpler.

5.2 Fragments of LTL

Alur and La Torre, in their paper [14], consider some restrictions on LTL syntax. Such restrictions are called *fragments*. In particular, they consider the following two:

1. LTL(F, X, $\wedge$, $\vee$): only formulas obtained using the operators $\mathrm{Finally}$, Next, AND and OR are allowed;

2. LTL(G, F, $\wedge$, $\vee$): only formulas obtained using operators Always, $\mathrm{Finally}$, AND and OR are allowed.

We consider the set of LTL(F, X, $\wedge$, $\vee$) formulas ϕ, and, for each of them, the language $L(\phi)$ consisting in computations which satisfy ϕ. They prove the following theorem for LTL(F, X, $\wedge$, $\vee$) formulas:

Theorem 5 (**on the determination of automata accepting LTL(**F**,** X**,** $\wedge$**,** $\vee$**) formulas**)**.** *For every LTL(F, X,* $\wedge$*,* $\vee$*) formula* ϕ *exists a deterministic Büchi automaton* A_ϕ*, doubly exponential in the length of the formula, which recognizes the language* $L(\phi)$*.*

Theorem 5 tells us that solving LTL(F, X, $\wedge$, $\vee$) games is simpler than solving general LTL games (we leave the discussion on complexity as an open problem). Since LTL(F, X, $\wedge$, $\vee$) contains combinations of safety and guarantee properties of great practical value, this simplification constitutes an important theoretical result. We can define a fragment of LTL[Z] analogous to LTL(F, X, $\wedge$, $\vee$), which we call LTL[Z](F, X, $\wedge$, $\vee$), allowing for formulas ϕ just the use of the operators F, X, $\wedge$ and $\vee$, interpreted of course according to Zadeh's semantics. If we can extend Theorem 5 to LTL[Z](F, X, $\wedge$, $\vee$) formulas, we would obtain a great simplification for the corresponding LTL[Z] games, too. In the next section we prove that such an extension is possible.

5.3 Fragments of LTL[Z]

To prove Theorem 5 for LTL[Z](F, X, $\wedge$, $\vee$) formulas we need the following lemma, stated and proven in [15]:

Lemma 2 (booleanization of fuzzy LTL formulas). *For every fuzzy LTL formula ϕ and every interval $P \subseteq [0,1]$, exists a Boolean LTL formula, which we denote with $Bool(\phi, P)$, such that, for every path π on a given Kripke structure, $[\![\pi, Bool(\phi, P)]\!] = 1$ if and only if $[\![\pi, \phi]\!] \in P$.*

The proof that Almagor, Boker and Kupferman give of this lemma is constructive, i.e., they construct the formula $Bool(\phi, P)$ and prove it has the property asserted in the statement of the lemma. This constructions consists in translating every fuzzy LTL operator which appears in ϕ into a precise combination of LTL operators. Their proof is valid for a more general logic. We only report the part of the construction which regards LTL(F, X, $\wedge$, $\vee$). We expose the construction referencing explicitly to LTL[Z]:

- constant $True$:

$$Bool(True, P) = \begin{cases} True \text{ if } 1 \in P \\ False \text{ if } 1 \notin P \end{cases} \tag{5.1}$$

- constant $False$:

$$Bool(False, P) = \begin{cases} True \text{ if } 0 \in P \\ False \text{ if } 0 \notin P \end{cases} \tag{5.2}$$

- atomic propositions:

$$Bool(p, P) = \begin{cases} True \text{ if } 0 \in P \text{ and } 1 \in P \\ p \text{ if } 0 \notin P \text{ and } 1 \in P \\ \neg p \text{ if } 0 \in P \text{ and } 1 \notin P \\ False \text{ if } 0 \notin P \text{ and } 1 \notin P \end{cases} \tag{5.3}$$

- operator AND:

$$Bool(\phi \wedge \psi, P) = \vee_{d_1 \in V(\phi), d_2 \in V(\psi) : d_1 \wedge d_2 \in P} Bool(\phi, d_1) \wedge Bool(\psi, d_2); \tag{5.4}$$

- operator OR:

$$Bool(\phi \vee \psi, P) = \vee_{d_1 \in V(\phi), d_2 \in V(\psi) : d_1 \wedge d_2 \in P} Bool(\phi, d_1) \vee Bool(\psi, d_2); \tag{5.5}$$

- operator Next: we set $Bool(X\phi, P) = X(Bool(\phi, P))$;

- operator $\mathrm{Finally}$: it is written in terms of operator $Next$.

In the expressions for operators AND and OR, $V(\phi)$ and $V(\psi)$ represent the sets of all possible values formulas ϕ and ψ respectively can have, which are finite according to Lemma 1. In these expressions for operators F, X, $\wedge$ and $\vee$ we see that only operators F, X, $\wedge$ and $\vee$ appear. It means that an LTL[Z](F, X, $\wedge$, $\vee$) formula is translated, following this constructions, into an LTL(F, X, $\wedge$, $\vee$) formula. This is a crucial point in the proof of the theorem we want to establish for LTL[Z](F, X, $\wedge$, $\vee$), which we now state and prove as follows:

Theorem 6 (on the determination of automata accepting LTL[Z](F, X, $\wedge$, $\vee$) formulas). *For every LTL[Z](F, N, $\wedge$, $\vee$) formula ϕ and interval $P \subseteq [0, 1]$ exists a deterministic Büchi automaton A of exponential size which accepts all and only computations satisfying ϕ.*

Proof. Given the LTL[Z](F, X, $\wedge$, $\vee$) formula ϕ and the interval $P \subseteq [0, 1]$, let us construct the LTL formula $Bool(\phi, P)$ applying Lemma 2. Analyzing the construction at the basis of the lemma, we have seen that $Bool(\phi, P)$ is an LTL(F, N, $\wedge$, $\vee$) formula. We can therefore apply Theorem 5 to it, and construct a deterministic Büchi automaton $A_{Bool(\phi,P)}$ which accepts all and only those computations which satisfy $Bool(\phi, P)$. But, by construction, a computation π satisfies $Bool(\phi, P)$, if and only if $[\![\pi, \phi]\!] \in P$. $A_{Bool(\phi,P)}$ is therefore the deterministic Büchi automaton we are looking for. $\square$

Apart the practical importance of this theorem, we note that it also implies a remarkable theoretical result: indeed, it tells that LTL[Z](F, X, $\wedge$, $\vee$) games are almost [1] as easy to solve as LTL(F, X, $\wedge$, $\vee$) ones. Their greater expressiveness, so to say, comes therefore at a little cost.

[1]Of course, we must consider that in order to solve LTL[Z](F, X, $\wedge$, $\vee$) games we have to construct the LTL formula $Bool(\phi, P)$, which requires an additional computational effort. Almagor, Boker and Kupferman prove that the translation can indeed be quite expensive in computational terms. Nonetheless, its cost should generally be small when compared to that required by the determinization of a Büchi automaton.

Conclusions

In this section we briefly summarize the work we have presented. Subsequently, we point out some issues worthy of further investigations, and give some suggestions about the possible developments of the theory.

Brief survey

In Chapter 1 we introduced the basic concepts of fuzzy control theory and fuzzy logic. In Chapter 2 we illustrated the suitability of temporal logic, and LTL in particular, to state and check properties of systems, and suggested the possibility to apply these concepts to the design of a new type of fuzzy controller. In Chapter 3 we went deeper into this topic, and also gave our main technical contribution: we defined LTL[Z] games and adapted the techniques known for classic LTL games to prove that solving them is decidable. In Chapter 4 we exemplified how to use fuzzy LTL games to modelize and solve control tasks, which is the main conceptual contribution of this work. Finally, in Chapter 5, we focused our attention on one of the most demanding points of the procedure to solve fuzzy LTL games, showing that it could be greatly simplified introducing restrictions on the fuzzy LTL formulas. Again, the result was obtained by combining previous results on classic LTL and fuzzy LTL. From this brief summary it is possible to recognize many points where we could have taken different directions, which could be investigated in subsequent works.

Possible future developments

We just suggest three lines of investigation, which we think would be of interest:

1. As we said, many different ways to fuzzify the Boolean and the temporal operators are possible. They lead to fuzzy versions of LTL different from LTL[Z], and therefore to different fuzzy LTL games. To the most common of them the results contained in [15] are still applicable, so, at least for what regards the theoretical part, nothing new shows up. Nonetheless, the difference in the semantics would lead to a different behavior of the fuzzy controller eventually implemented using that fuzzy LTL game as a model, making it more or less efficient. A study of which logic would be better in relation to the different systems to be controlled, or the different specifications to be fulfilled, would be interesting.

2. Two-player games are very useful to model the interaction between a system and its environment, but, in many cases, it is more realistic to consider the interaction between the environment and a greater number of systems, each with its specification (it is the field of study of multi-agent systems' theory). The possibility to define multi-player LTL and LTL[Z] games, and to apply them to design multi-agent fuzzy controllers, could be worth studying.

3. It is possible to try to implement the algorithm developed in Chapter 3, in order to make simulations and test its efficiency.

Further readings and related works

During the preliminary phase of the writing of this thesis a bibliographical research has been done, to establish what was the state of the art for what concerns the main issues here investigated: fuzzifying temporal logic and designing game-based fuzzy controllers. Though we preferred to introduce the new ideas previously exposed, we report in this section what we think are among the most interesting attempts done in the aforementioned subjects, to acknowledge them as an inspiration for our work and to compare them with the path we have chosen.

Other attempts to fuzzify LTL

The way we have chosen to fuzzify LTL consists in allowing fuzzy values for the atomic propositions, and in adapting LTL semantics to deal with such values. In this way we have been able to talk about the degree of satisfaction of LTL formulas, or, as is said in [15], about the *quality* of satisfaction of these formulas. More precisely, in [15] this concept is called *propositional quality*, since the evaluation of the quality of the LTL formulas is intended with respect to propositions. The authors also refer to *temporal quality* as the framework where the degree of quality of an LTL formula is evaluated with respect to the temporal operators. In particular, they introduce *discounting functions* $\eta : \mathbb{N} \to [0, 1]$, which are monotonically decreasing and tends to 0, and then add to the ordinary $Until$ operator a *discounted Until* for every discounted function, whose semantics is defined this way:

$$[\![\pi, \psi_1\ U_\eta\ \psi_2]\!] = \sup_{i \geq 0} \left\{ \min \left\{ \eta(i)[\![\pi_{\mathrm{i}}, \psi_2]\!], \min_{0 \leq j < i} \eta(j)[\![\pi_{\mathrm{j}}, \psi_1]\!] \right\} \right\} \tag{5.6}$$

(operators F_η and G_η are obtained from U_η). Basically, since the function η is decreasing, here the states nearer in time count more than the farther ones, allowing one to give specifications whose quality evaluation depends not only on the states visited along the path, but also on the time elapsed while waiting for the fulfillment of eventualities. Different definitions of temporal quality can be found in other works. In [18], for example, *bounded* versions of the temporal operators are introduced. They tolerate that the specification is not fulfilled for a certain number of instants, a penalty being paid for every of such instant, as prescribed by appropriate *avoiding functions*. Temporal quality seems indeed a fertile field of investigation.

System-environment models using games different from LTL

With respect to the possibility to describe the interaction between a system and the environment as a game, [19] is a work where an interesting approach, different from the one we have chosen, is described. In this paper, the authors introduce what they call *Fuzzy Game Graphs (FGG)*. These are basically Kripke structures, partitioned as for LTL games, with fuzzy values on the transitions to represent the possibility of success of a determined move. The winning condition is of the reachability type, i.e., Player 0 has to reach a particular region of the arena, called the winning region, and Player 1 must impede her. This game is different from LTL games in two aspects, both in the structure of the arena and in the winning condition (the last is not a great difference though, since reachability can be expressed in LTL). In an example at the beginning of their work, the authors use an FGG to describe the interaction between a robot and a hostile environment, in what can be considered the model for the design of a fuzzy controller. The main difference with respect to our case is that fuzzy values, here, represent possibilities rather than degrees of truth. Nonetheless, their approach has been very influential on our work, and we must acknowledge them also for the concept of value of a game, which we have adapted to LTL[Z] games.

Appendix A

Constructing the Vardi-Wolper Büchi automaton

It is remarkable that the results proven in this work completely rely on the standard techniques usually applied to solve the model-checking problem for LTL formulas and LTL games. For this reason, no fundamentally new construction has been needed, and it has been possible to omit the technical details almost completely. However, it seems inappropriate to conclude this work without giving an insight into the automata-theoretic techniques which play such an important role in this field. For this reason, we now show how, from an LTL formula ϕ, the nondeterministic Büchi automaton A_ϕ introduced in Subsection 3.2.2 can be constructed. We will follow [20].

1. We begin with a preliminary definition. We define the *closure* of formula ϕ, $cl(\phi)$, as the set of ϕ's sub-formulas having the following properties:

   - $\phi \in cl(\phi)$,
   - if $\phi_1 \wedge \phi_2 \in cl(\phi)$, then $\phi_1 \in cl(\phi)$ and $\phi_2 \in cl(\phi)$,

- if $\phi_1 \vee \phi_2 \in cl(\phi)$, then $\phi_1 \in cl(\phi)$ and $\phi_2 \in cl(\phi)$,
- if $X\ \phi_1 \in cl(\phi)$, then $\phi_1 \in cl(\phi)$,
- if $\phi_1\ U\ \phi_2 \in cl(\phi)$, then $\phi_1 \in cl(\phi)$ and $\phi_2 \in cl(\phi)$;

now we can start constructing the automaton:

2. the alphabet Σ is the set 2^P, where P is the set of atomic propositions;
3. the set of states Q is formed by all the possible elements s of $2^{cl(\phi)}$ such that:
    - $False \notin s$,
    - if $\phi_1 \wedge \phi_2 \in s$, then $\phi_1 \in s$ and $\phi_2 \in s$,
    - if $\phi_1 \vee \phi_2 \in s$, then $\phi_1 \in s$ or $\phi_2 \in s$;
4. given a state s and a word a, we have that transition t belongs to $\delta(s, a)$ if and only if:
    - for every $p \in P$, if $p \in s$, then $p \in a$,
    - for every $p \in P$, if $\neg p \in s$, then $p \notin a$,
    - if $X\ \phi_1 \in s$, then $\phi_1 \in t$,
    - if $\phi_1\ U\ \phi_2 \in s$, then either $\phi_2 \in s$, or $\phi_1 \in s$ and $\phi_1\ U\ \phi_2 \in t$;
5. the initial states are those which contain ϕ;
6. be $e_1, e_2, ..., e_k$ the set of all eventualities in $cl(\phi)$, i.e., the set of all the expressions of the type $\phi_1\ U\ \phi_2 \in cl(\phi)$; a state s_i is accepting if and only if $e_i, \phi_2 \in s_i$, or $e_i \notin s_i$.

We do not prove the correctness of the construction, remanding for this to [20]. We only wanted to illustrate how the translation from LTL to automata is done, being it a crucial point for every work in the present theory of verification.

Bibliography

[1] Dorf R. C., Bishop R. H. (2011) *Modern Control Theory*, *Prentice Hall*, 12th edition.

[2] Maxwell J. C. (1868) *On Governors*, *Proceedings of the Royal Society*, 100.

[3] Zadeh L. A. (1965) *Fuzzy Sets*, *Information and Control*, 8: 338–353

[4] Pollack A. (1989) *Fuzzy Computer Theory: How to Mimic the Mind?*, *The New York Times*, Retrieved 2011-03-11.

[5] https://www.mitsubishi-termal.it/en/mitsubishi-air-conditioning/, visited on October 7th, 2019.

[6] http://www.freepatentsonline.com/5251358.html, visited on October 7th, 2019.

[7] Passino K. M., Yurkovich S., *Fuzzy Control*, *Addison-Wesley*.

[8] Ramadge P. J., Wonham W. M. (1987) *Supervisory Control of a Class of Discrete Event Processes*, *Control and Optimization*.

[9] Pnueli A. (1977) *The Temporal Logic of Programs*, *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS)*, 46–57.

[10] Piterman N. (2007) *From Nondeterministic Büchi and Streett Automata to Deterministic Parity Automata*, *Ecole Polytechnique Fédéral de Lausanne (EPFL).*

[11] Vardi M. Y., Wolper P., (1994) *Reasoning about infinite computations*, *Information and Computation*, 1-37.

[12] Krishnan S. C. (1998) *ω-automata, games, and synthesis.*

[13] McNaughton R. (1993) *Infinite games played on finite graphs*, *Annals of Pure and Applied Logic*, 65, 149-184.

[14] Alur R., La Torre S. (2001) *Deterministic Generators and Games for LTL Fragments*, *Proceedings of the 16th Annual IEEE Symposium on Logic in Computer Science*, 291-300.

[15] Almagor S., Boker U., Kupferman O. (2016) *Formally Reasoning About Quality*, *J. ACM*, 63.

[16] Pnueli A., Rosner R. (1989) *On the Synthesis of a Reactive Module.*

[17] Roggenbach M. (2001) *Determinization of Büchi-Automata*, *Lecture Notes in Computer Science.*

[18] Frigeri A., Pasquale L., Spoletini P. (2012) *Fuzzy Time in LTL.*

[19] Pan H., Li Y., Cao Y., Li D. (2017) *Reachability in Fuzzy Game Graphs*, *IEEE Transactions on Fuzzy Systems*

[20] Wolper P., *Constructing Automata from Temporal Logic Formulas: A Tutorial.*