Université Sidi Mohammed Ben Abdellah
Faculté des Sciences Dhar El Mahraz- Fès
Centre d'Etudes Doctorales
"Sciences et Technologies"

Formation Doctorale : STIC

Discipline : Sciences et Techniques

Spécialité : Informatique& Modélisation

Laboratoire : Laboratoire d'Informatique, Modélisation et Systèmes

THESE DE DOCTORAT

Présentée par

OUCHEIKH Rachid

Real-time Systems Verification and Graph Theory

Soutenue le Lundi 16 Avril 2018, devant le jury composé de :

Pr. Mohammed Ouçamah Cherkaoui Malki	***Faculté des Sciences Dhar EL Mahraz, Fès***	***Président***
Pr. Lahcen Omari	***Faculté des Sciences Dhar El Mahraz, Fès***	***Directeur de thèse***
Pr. Mourad El Yadari	***Ecole Supérieure de Technologies, Mèknes***	***Rapporteur***
Pr. Mohammed Al Achhab	***Ecole Nationale des Sciences Appliquées, Tétouan***	***Rapporteur***
Pr. Saad Dosse Bennani	***Ecole Nationale des Sciences Appliquées, Fès***	**Rapporteur**
Pr. Ismail Berrada	***Faculté des Sciences Dhar EL Mahraz, Fès***	**Co-encadrant**
Pr. Mohamed El Kamili	***Faculté des Sciences Dhar EL Mahraz, Fès***	**Examinateur**

Année universitaire : 2017-2018

Abstract

The software component is paramount in most critical systems and involves system security and its proper operation. In fact, software errors can cause enormous damage, financial crises and even human deaths. Therefore, system verification is a primary task in their design process. We are interested in parametric real-time systems that have timing constraints as their main feature. In this thesis, we build a new techniques for verification based on Parametric Timed Automata (PTA) as modeling tool, and Constraint Satisfaction Problems (CSPs) formalism as computation method for timing constraints resolution. CSPs present an efficient tool for expressing and solving a variety of real problems. Once the problem is expressed as a set of finite constraints, the deal is to find the states and variables values satisfying them. Even though the problem is, in general, NP-complete, there are some approximation and practical techniques to tackle its intractability. The most used one is the Constraint Propagation. It is about a reasoning process which consists in explicitly excluding values or combinations of values for some variables of a problem whenever they make a given subset of its constraints not satisfied.

In this thesis, we deal with a CSP subclass, called 4-CSP, for which the constraint network (CN) is comprised of relations of the form: $\{x \sim \alpha, x - y \sim \beta, (x - y) - (z - t) \sim \lambda\}$, where x, y, z and t are real variables, α, β and λ are real constants and $\sim \in \{\leq, \geq\}$. We provide the first graph-based proofs of the 4-CSP tractability, and we elaborate a 4-CSP resolution algorithm based on the closure and the positive linear dependence theories coupled with the constraint propagation technique. The time and space complexities of resolution algorithms are shown to be polynomial and the comparison with other works is discussed.

Finally, we show the importance of these techniques in many applications such as parametric real-time verification and static code analysis. Mainly, we use this computation power to track the tractability of parametric timing constraints of systems modeled by PTA, providing thereby the forward and backward reachability analysis.

Keywords: 4-Octahedron abstract domain, real-time systems verification, model-checking, Constraint Satisfaction Problem, code static analysis, canonical form, Galois connection, Hyper-graph, positive linear dependence, reachability analysis.

Résumé

La composante logiciel est primordial dans la plupart des systèmes critiques et met en jeu la sécurité des systèmes et leur bon fonctionnement. En fait, les erreurs logicielles peuvent causer d'énormes dégats, des crises financières et même des pertes humaines. Par conséquent, la vérification des systèmes est la tâche principale dans leur processus de conception. Nous nous intéressons aux systèmes temps-réel paramétriques qui ont les contraintes de temps comme caractéristique principale. Dans cette thèse, nous construisons de nouvelles techniques de vérification basées sur des Automates Temporisés Paramétrés (ATP) comme outil de modélisation, et le Problème de Satisfaction de Contraintes (PSC) formalisme comme cadre de calcul pour la tractabilité des contraintes temporelles. Les CSP présentent un cadre pour la modélisation et la résolution d'une variété de problèmes réels. Une fois que le problème est exprimé en un ensemble de contraintes finies, le défi est de trouver les états et les valeurs des variables qui les satisfont. Même si le problème est, en général, NP-complet, il existe des approximations et des techniques pratiques pour faire face à son caractère intraitable, la plus utilisée étant la propagation des contraintes. Il s'agit d'un processus de raisonnement qui consiste à exclure explicitement des valeurs ou des combinaisons de valeurs pour certaines variables d'un problème à chaque fois qu'elles rendent un sous-ensemble donné de ses contraintes non satisfait.

Dans cette thèse, nous traitons une sous-classe de PSC, appelée "4-CSP", pour laquelle le réseau de contraintes inclue des relations de la forme : $\{x \sim \alpha, x-y \sim \beta, (x-y)-(z-t) \sim \lambda\}$, où x, y, z et t sont des variables réelles, α, β et λ sont des constantes réelles et $\sim \in \{\leq, \geq\}$. Nous fournissons les premières preuves de la traitabilité de 4-CSP, et nous élaborons un algorithme de résolution de 4-CSP basé sur les théories de fermeture et de dépendance linéaire positive combinées à la technique de propagation des contraintes. La complexité temporelle et spatiale

des algorithmes de résolution est démontrée être polynomiale et la comparaison avec d'autres travaux est bien discutée.

Enfin, nous montrons l'importance de ces techniques dans de nombreuses applications telles que la vérification des systèmes temps-réel paramétriques et l'analyse statique de code. Principalement, nous utilisons cette puissance de calcul pour étudier la tractabilité des contraintes paramétrées de temps pour les systèmes modélisés par ATP, fournissant ainsi les algorithmes de l'analyse d'accessibilité en avant et en arrière.

Mots-clés : Domaine abstrait 4-Octahedron, Vérification des systèmes temps-réel, model-checking, Problème de satisfaction des contraintes, Analyse statique de code, Forme canonique, Connexion de Galois, Hyper-graphes, Dépendance linéaire positive, Analyse d'accessibilité.

Acknowledgment

I would like to thank Prof. Mohammed Ouçamah Cherkaoui Malki, Prof. Saad Dosse Bennani, Prof. Mohamed Al Achhab, Prof. Mourad El Yadari, Prof. Mohamed Elkamili and Prof. Lahcen Omari for agreeing to be part of my jury.

I would like to thank very warmly Prof. Ismail Berrada, my thesis co-director, who guided me for several years. I thank him for all the attention he paid to my work, for his confidence, his patience, his constructive requirements and his moral support. He is a great man !

To my mother, my wife and my little girl I dedicate this work.

I also want to thank the "National Center for Scientific and Technical Research" (CNRST) for funding my thesis. It was very helpful for the conduct my researches.

I thank all those who helped me from far or near to succeed in my life !

Table des matières

Chapter 1

Introduction

1.1 The Stakes

Today, a software project is often still managed in the same way as the construction of castles in the last centuries. The increasing complexity of computer applications unfortunately has the corollary of the existence of abnormal behavior leading sometimes to irreparable consequences. We read and we still read stories reporting catastrophic and disastrous software errors. Among of these, we can mention the failures of the medical device Therac-25 which, between 1985 and 1987, caused the death of several people as a result of massive radiation overdoses. A detailed study of the causes of these accidents came to the conclusion that these errors were due to software problems [70].

Another dramatic disaster is the failure of the first rocket Ariane 5 in 1996, which is due to the total loss of guidance and attitude information 37 seconds after starting the main engine ignition sequence. On June 4, 1996, the explosion of this rocket caused the failure of the mission and a severe loss of 700 million euros for the Ariane Espace. The independent investigation group came to the following conclusions [89]:

The Ariane 5 rocket contains two inertial reference systems (IRS). On its first flight, the two IRS declared a failure due to an exception triggered when converting a floating-point number to a unsigned integer.The conversion was not protected and the floating number was greater than the largest machine-readable integer. It therefore produced a non-significant

number that was interpreted by the flight computer as the altitude of the rocket. The computer reacted by ordering the engines to straighten the rocket, causing its deviation and then its explosion.

The most interesting issue of this bug is that the conversion error occurred because the values provided by the sensors were higher than expected. This loss of information is due to errors in the specification and design of the inertial reference system software, as reported by the inquiry board [89].

During the first Gulf War, the US military used many Patriot missiles to protect the strategic points against the Iraqi Scud missiles. The Patriot missile is a fully automatic ground-to-air missile that searches for and intercepts enemy craft in a certain area of space. On the night of February 25, 1991, one of the Patriot missiles launched from Dhahran, Saudi Arabia, failed to intercept a Scud missile, resulting in the death of 28 US soldiers.

To predict the trajectory of the missile, the system uses two values: the speed of the missile measured by the radar and the time spent by the radar for its detection. The time is encoded by an integer expressing the number of tenths of seconds since the last reboot of the system. This integer is then converted to a floating 24-bit value to calculate the missile trajectory. When using the Patriot system for a long time, the integer representing the time becomes very important and the conversion to a floating value causes loss of precision which directly affects the decision process of trajectory estimation. On February 25, 1991, the Patriot missile battery had been operating for 100 hours, the error in the prediction of the trajectory was 687 meters and the missile was not intercepted. The official investigation conducted by the American army [51] has come to the following conclusion: “the loss of precision when converting an integer to a 24-bit floating number is therefore the only cause of the Patriot Missile failure”.

We can also cite the USS Yorktown blocking in September 1997. The USS Yorktown ship is a missile cruiser of the American army. The cause of this incident was a misuse of a database remote management program. Indeed, the operator responsible for entering the collected data in the database accidentally typed the number 0. This resulted in a division by

0 in the program, and consequently a crash of the underlying operating system. In addition to these relevant examples, we cite also the breakdown of the telephone network in the USA in 1989, the recall of hundreds of thousands of Pentium machines by INTEL in 1996, the breakdown of France Telecom's mobile network Paris region in January 98, etc.

These examples are just illustrations among many other critical systems where the software component is paramount. In such contexts, systems need to be responsive. Other applications of these systems is more critical. In fact, the production of chemical elements such as ethylene or the prophyline is thus managed and controlled by computers. These systems require a high accuracy in the time of action. The slightest mistake could be the source of an ecological disaster. To fully realize the difficulty of the problem, one should know that software invade all objects of our everyday life (almost everything is controlled by code fragment) and that some major software has tens of millions of instructions (tens of millions in a mobile phone). The omnipresence of computers in our daily lives (databases, control / command systems, transport technologies, telecommunications,...) is an evidence. In addition, with the evolution of computer technology, machines are faster and have large memory, which enable them to run bigger software. This latter becomes more and more complex and very difficult to design (conception) without error. Moreover, their design obeys to methodological principles that are still rudimentary.

The error is often forbidden, as in the case of many fields such as aeronautics, telesurgery, telepayment, military applications, or the control of a nuclear power station. The software is then said to be critical. In addition, as noticed in all the cited cases, a program error had a very strong effect (financial and environmental harm for Ariane 5 and USS Yorktown, human loss for the Patriot missile) because of the interaction between this program and the sensors and/or actuators allowing the communication with the outside world. It is therefore essential, before using programs in embedded systems, to ensure they operate correctly.

It becomes, then, crucial to have methods to certify software in which a failure can have dramatic consequences. All these facts highlight the need to prevent computer bugs. Critical software verification is a fast-growing field of research that relies on a recent, but

booming, industrial demand. In most cases, these critical software aim to control a process that communicates with its environment via sensors, thermometers, signals, keyboards, etc. They are not intended to calculate a result, but to ensure the continuous functioning of the controlled process, that is why they are called reactive systems. Another common feature of these critical systems is that their overall behavior depends on the interaction of several subsystems evolving in parallel, they are distributed systems. In addition, the time parameter usually intervenes explicitly, hence, they are called real-time systems.

Whether critical or not, it is on the specification and verification of distributed real-time reactive systems that we are going to focus. In fact, the cost and risk involved in their design leads to the need for a correct and robust modeling of the system before its deployment. Therefore, the release of software on the market assumes of course a costly phase of testing. However, even if this phase proved to be able to identify a good part of the errors, it is in no way a guarantee.

1.2 Formal Methods

We have affirmed that another reason for the presence of errors in software is the complexity of the programs that current machines can execute. But we should not forget that one of the error sources in software lies in the fact that in most cases the people who take care of the drafting of the specifications and the software specification are not the people who program them. Since the specification is very often written in natural language, it contains ambiguities, inaccuracies or even omissions or contradictions.

There are several methods for checking the correct operation of a machine (software). Traditionally, this is accomplished by methods like reviewing of design documents and source code and by extensive simulation and testing of the system and its components. There are, in general, two ways to accomplish these missions: the proof methods, where the engineer uses a series of proofs to prove manually that the studied system meets its specifications; and the model-checking which aims at proving automatically some properties (mostly of safety) or their negation.

The first preferred approach for this mission is to test the programs by running them

on examples (called scenarios), for different time length and in various environments. It is noticed that this process is often time-consuming and provides only statistical measures of correctness. To evaluate this method, we can say that it can not generally be exhaustive and it is rarely possible to test all possible cases. Nevertheless, the test still remains an essential component of software development as it allows to discover errors.

By following this logic, it seems crucial to have rigorous design methods and automatic and efficient validation techniques. These must be done as the system development progresses. Indeed, more an error will be detected later, more its risks are expensive. Moreover, such a work can only be done in a well-defined mathematical framework to avoid all problems arising from non-formal language, non-globalist tests, or non-rigorous suggestions.

The importance of formal methods lies in their ability to provide a mathematical framework for thorough and strict systems and programs description. This formal framework makes it possible to remove ambiguities existing in terms of specifications and natural language. They therefore aim to provide a correct and holistic specification.

In the following paragraphs, we recall the most used formal methods, and we mention their specificities.

1.2.1 Test generation

The test sequence generation techniques consist in computing, from a formal specification of the system and the property to be tested, a set of interesting "scenarios" to increase the confidence that one can have towards the behavior of the software. In other words, it consists of generating, automatically or manually, relevant test scenarios on the system already developed in order to detect errors and compare the test outputs with the desired system behavior.

Its objective is to minimize the chances of occurrence of an anomaly with automatic or manual means. It aims to detect at once the various possible anomalies and the possible defects that could cause them. Indeed, testing consists in trying to find errors or gain some confidence in the correctness of an implementation with respect to a specification by the execution of test cases.

There are two main types of test: the structural test and the functional test. The struc-

tural test (White-box) is based on the analysis of the internal structure of an implantation while the functional test (Black-box) consists of checking whether a software or hardware layout complies with its specification, without any knowledge of internal implementation and without seeing the source code. The latter is defined as a description of desired behaviors that describes what the system should do and not how it is done.

Testing is applied in several domains and a lot of theoretical works have been done on test generation algorithms [83]. This technique is insufficient because it is not exhaustive and it proves the existence of errors, but not their absence. If the system is unknown, test cases can be generated to interact with it and detect any nonconformities with the specification. However, if the system is known, verification can be done at different levels of abstraction. Programs can be checked and there are certified compilers for extremely critical cases. The material can also be verified by simulation.

1.2.2 The automatic demonstration

This technique allows, from a system and a property expressed in the same specification language, to prove that the property is verified by the system or not. This is done using deduction rules, as one could do to show a mathematical theorem. A demonstrator capable to make such a proof for each system and each property is of course impossible to build, but, currently, "assistants to the proof" make it possible to write proofs. The user has to give axioms to the tool, and, sometimes, indications to help.

Automatic proof demonstrators are used commercially primarily in the design and verification of integrated circuits. Since the problem of the FDIV bug of the Pentium processor, the design of the floating computing unit microprocessors is the subject of increased care. AMD, Intel, and other manufacturers use automated theorem provers to verify that some arithmetic operations are properly implemented in their processors.

This method is exhaustive, but it is very difficult to implement.

1.2.3 Model-checking

The common procedure in all these so called formal methods is to define the system under development in a formal framework and then apply rigorous methods within this framework to prove that the system meets its requirements. However, due to the complexity of real-life systems, applying formal methods is often considered too difficult. A way to facilitate the mission is to automate the analysis, for instance by using model checking. In contrast to manual techniques, model-checking is fully automatic in the sense that the proof showing that a system satisfies a given requirement is constructed by the model-checker without manual interaction. It is then an automatic procedure that allows to verify algorithmically whether a given model, the system itself or an abstraction of the system, satisfies a logical specification, usually formulated in terms of temporal logic.

1.3 Problem statement

In this thesis, we study and develop formalisms for the analysis of systems in which time is a critical and important factor. We track the tractability of constraints appearing in the parameterized timed systems.

1.4 Research contributions

Our research is divided into two shutters: theoretical part and practical part. The contributions of each part consist of the following:

Theoretical outcomes

1. Setting a theoretical basis for the 4-Constraint Satisfaction Problem and giving a data structure for its domains.

2. This method is based on hypergraph theory coupled with positive linear dependence theory.

3. Developping a modified arc consistency algorithm that combines the MAC algorithm with the hypergraph closure in order to solve easily the 4-CSPs: either to check the emptiness of solution set or to list the solutions. All these operations have a polynomial time and space complexity.

Practical contributions

The practical contributions are divided into two application parts:

In the real-time system verification part, we aim to reach these finalities:

1. Application of these outcomes to the verification of real-time systems modeled by the parametric timed automata.

2. Track the undecidability of a subclass of parametric timed automata.

In the abstract interpretation field, we achieved these goals:

1. Introduction of a new abstract domain built on the 4-CSP framework. The precision of our domain lies between the octagon domain [78, 101] which encodes the relation of the type: $\pm x \pm y \leq k$ for $k \in \mathbb{R}$ and the octahedra domain [39].

2. One of the scopes of this application is to explore how we can characterize the feasibility of polyhedra in general, and 4-octahedra in particular. We try to obtain similar results in graph theory able to characterize the feasibility of polyhedra as those of Bellman.

1.5 Structure of the Thesis

This thesis is organized in five chapters.

The **second chapter** presents the formalisms and theoretical basis of real-time system verification. It defines precisely the timed framework on which our work rely. Overall, the first section of this chapter presents the model of Timed Automata and Parametric Timed Automata (PTA) and states the essential properties of this model, the undecidability results and the algorithmic complexity. It then makes a state of the art of the work that has been done around the timed automata and the semantics of the clock languages. Thereafter, we

study at the end of this section the expressiveness of all the decidable classes that we have identified.

The second section of this chapter shed the light on the model-checking technique, how is it applied to the parametric timed automata. It recalls the different temporal logics and various ways to express (timed) properties that a system should verify in the model-checking procedure. Afterwards, it deals with the algorithmic aspect that explains the symbolic analysis of timed systems.

In the third section, we present the role of abstract interpretation in the software validation. We make plain the theoretical basis of abstract domain soundness before giving an example of the driver behavior analysis.

In **Chapter 3**, we examine the Constraint Satisfaction Problem (CSP) and their usefulness in resolving many real life issues. The first section is introductory, the second one presents the CSP language, and the third lists some techniques that was elaborated in order to resolve CSP and discusses their complexities.

In **Chapter 4**, we set up a theoretical basis for the 4-Constraint Satisfaction Problem. Indeed, we provide, based on hypergraph theory coupled with positive linear dependence theory, the first graph-based method for the 4-CSP tractability. Afterwards, in the second section we present some proven results on Positive Linear Dependence theory. The next two sections formalize the 4-CSP, give the hypergraph based characterization of its tractability and provide the computation algorithms. Finally, in order to benefit from these theoretical results, the fifth section deal with their implementation providing the 4-CSP data-structure and the canonical form computation algorithms.

In **Chapter 5**, we present some relevant applications of our elaborated techniques, such as Parametric Timed Systems Verification (section 5.1), Abstract Interpretation field (section 5.2) and Multimodal transport (section 5.3).

In the **last chapter**, we make a complete review of the work done in this thesis and we discuss some research perspectives, which present the continuity of the thesis.

Chapter 2

Verification of real-time systems

Introduction

The list of critical applications of real-time systems is long and provides many reasons to develop verification methods. This thesis work is situated in the frame of model-checking. The principle of model checking is described as follows: Suppose that we are given a system (in the form of a program or a specification for example) and a property of this system. The preliminary stage of model-checking consists on the one hand in modeling the system, that is to say constructing an object in a well-defined formal framework which reproduces as faithfully as possible the behavior of the real system. It consists secondly in modeling the property to be verified in a suitable specification language. Model-checking then consists in verifying that the model constructed for a system verifies the property expressed in the aforementioned language (hence the term "model-checking"). To achieve this, it is of course necessary to have model-checking algorithms.

The expected properties of a real-time system can be represented in many ways. Depending on the degree of criticality, knowledge of the system and properties to check, many approaches are possible. We have three complementary major research lines to improve the techniques of model-checking:

1. The **development of models** in order to properly represent the systems under study as well as the properties that one seeks to verify.

2. The **development of efficient algorithms** and tools taking into account the specificities of the chosen models.

3. The **validation of the techniques** developed by case studies.

The first work is to define the model classes, (for systems as well for properties), adapted to the problem considered. One of the model development criteria for model-checking is that this latter should be decidable, that is to say that it is possible to develop algorithms that "compute" whether or not the model of the system verifies the property model. The supported models are then the result of a compromise between an expressiveness allowing to describe many systems in a correct and natural way, a brevity allowing a synthetic description and avoidance of the combinatorial explosion problem and a simplicity making that the model-checking is at least decidable (with a reasonable complexity). The current chapter describes the modeling of timed systems since we are interested in the Parametric Timed Automata model.

2.1 Timed Automata model

Probably, the easiest way to describe the behavior of a system is to represent it by an automaton. An automaton is a graph containing labeled nodes and arcs. A node represents a possible state of the system and an arc (or transition) the activity linking two nodes: one is the state before the execution of the action and the other is the state reached after the action. Automata have been expanded in many ways and used for different purposes. The semantics of several specification languages is given in terms of automata. The use of automata provides a solid mathematical framework, eliminating confusion or ambiguity for system validation. Although it provides a pleasing graphical representation for system descriptions, this representation proves to be impractical for complex systems that include broad and detailed descriptions.

According to Hubert Garavel, a futuristic formalism should satisfy four essential criteria:

1. Expressiveness: The formalism must make it possible to model simply, without unnecessary contortions, the essential characteristics of asynchronous systems.

2. Universality: The formalism must be usable in all areas of activity involving asynchronous parallelism (software, hardware, telecommunications) and not be closely dependent on a particular domain (such as ESTELLE and SDL have been for telecommunications).

3. Executability: The formalism must have an executable character so that the modeling of the asynchronous systems do not serve only as documentation, but can be treated by tools of simulation (in order to effect the development), of rapid prototyping (to produce executable code) and automatic generation of tests.

4. Verifiability: Some asynchronous systems are sufficiently critical to the extent that their correction must be guaranteed by verification or proof techniques. However, these can only be applied to modeling made in formalisms whose semantics are rigorously defined and have good compositional and abstraction properties that make it possible to push back the limits of the explosion of states.

Automata satisfies the four aforementioned criteria. In fact, they are a fundamental modeling tool used to represent automatic devices, for instance reactive systems, as well as mathematical or physical objects . In this section, we investigate finite timed automata and parametric ones. We start with defining the Labeled Transition System (LTS) and Timed Automata (TA).

2.1.1 Automata

Any real system (for example a program) can be described by a system of transitions, i.e. any set of states and transitions labeled by actions between states. Being in one of the states of the transition system, it is possible to change state by performing an action that labels one of the outgoing transitions of that state.

Formally, a transition system is a pair $(S, \rightarrow)$ where S is a set of states and $\rightarrow$ is a set of state transitions (i.e. a subset of $S \times S$). A transition from state p to state q is written as $p \rightarrow q$. Neither the set of states is necessarily finite nor the set of transitions is necessarily finite. In addition, no "start" state or "final" states are given.

Let Σ be a finite alphabet, and write Σ^ϵ for $\Sigma \cup \{\epsilon\}$, where ϵ is the empty word ($\epsilon \notin \Sigma$). This alphabet presents the set of actions or events and ϵ designates the silent or internal action.

A labeled transition system is a tuple $(S, \Sigma, \rightarrow)$, where the set of alphabet Σ presents the labels. $(p, \lambda, q) \in \rightarrow$ is written as $p \xrightarrow{\lambda} q$. An execution in a system of transitions is then a sequence of actions. A sequence is a finite suite of actions that we simply note by juxtaposition: $\sigma = a_1 a_2 \dots a_n$ such that $a_i \in \Sigma$. The length of a sequence σ, denoted $|\sigma|$, is the number of actions that compose it. σ_i denotes the i^{th} action of σ (the index i is between 1 and $|\sigma|$). Now, we can define the automata.

Definition 1. *A Deterministic Finite-State Automaton (DFSA) is a tuple* $\mathcal{A} = (\Sigma, S, s_0, F, \lambda)$ *where*

- Σ *is finite alphabet of events,*
- S *is a finite set of control states,*
- $s_0 \subseteq S$ *is a privileged state, called initial control state,*
- $F \subseteq S$ *is a set of accepting control states, called also final states, and*
- $\lambda : S \times \Sigma \rightarrow S$ *is the transition function of* $\mathcal{A}$*. If* $\lambda(s, \sigma) = s'$ *such that* $s, s' \in S$ *and* $\sigma \in \Sigma$ *then the state* s' *is achieved from the state* s *after writing the alphabet* σ *(i.e. executing the event* σ*). The edge relating* s *to s' is labeled with* σ*;* $s \rightarrow s'$*.*

We usually draw an automaton, by displaying the states by circles, indicating the initial state by an incoming edge, the accepting states by a double circle or an outgoing arrow, and the transition from state s to state s' reading the letter a with an arc going from s to s' and labeled by a. The figure 1 illustrates an example of having four states, the initial state is $S0$ and the accepting one is $S4$. Its set of alphabets is $\Sigma = a, b, c, d, e, f$.

We naturally extend the transition function λ to $S \times \Sigma^*$ as follows: $\lambda(s, \epsilon) = s$ et $\lambda(s, aw) = \lambda(\lambda(s, a), w), a \in \Sigma, w \in \Sigma^\epsilon$. An automaton is deterministic if it has a single initial state

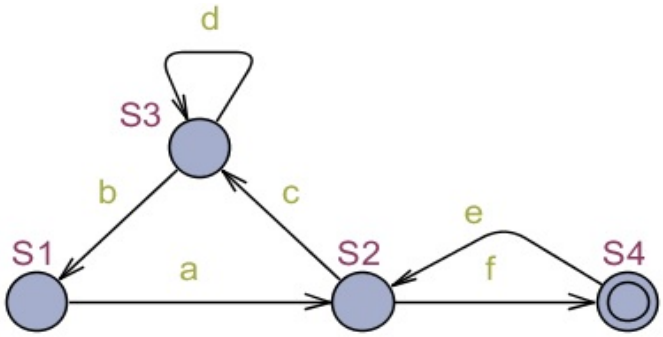

Figure 2.1: Example of Deterministic Finite-State Automaton

and if, for each of its states, there is at most one transition for each possible label. If it has exactly one transition by label, then we speak of a complete deterministic automaton. The accepting language of the automaton is then: $\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^* | \lambda(s_0, w) \in F\}$.

The non-deterministic automaton model generalizes the case of finite deterministic automaton. In fact, it turns out to be useful in several practical cases ((give examples)).

Definition 2. *A Non-Deterministic Finite Automaton (NFA) is a quintuple* $\mathcal{A} = (\Sigma, S, S_I, F, E)$ *where*

- Σ *is finite alphabet of events,*
- S *is a finite set of control states,*
- $S_I \subseteq S$ *is the set of initial control states,*
- $F \subseteq S$ *is a set of accepting control states, and*
- $E \subseteq S \times \Sigma^\epsilon \times S$ *is a transition relation. It defines the edge* (s, w, s') *that allows* w*-labeled transition from* s *to* s'*.*

It is easy to notice several differences between FDA and NFA. In the non-deterministic case, it is possible to have more than one initial state. In addition, edge labels are not necessary letters, they can be words of Σ^*. Finally, we no longer have a transition function but a transition (trinary) relation.

A word $w = a_1...a_k$ is accepted by an NFA $\mathcal{A} = (S, S_I, F, \Sigma, E)$ if it exists $s_0 \in S_I, l \in \mathbb{N} \setminus \{0\}, a_1, ..., a_l \in \Sigma^\epsilon, s_1, ..., s_l \in S$ such that: $(s_0, a_1, s_1), (s_1, a_2, s_2), ..., (s_{l-1}, a_l, s_l) \in E$,

$w = a_1, ..., a_l$ and $q_l \in F$. In other words, a word w is accepted by NFA if there is a path in the associated automaton graph labeled with w, starting in an initial state and ending in a final state. Obviously, the accepted language of a NFA $\mathcal{A}$ is the set of its accepted words, and it is noted $\mathcal{L}(\mathcal{A})$. Indeed, two NFA $\mathcal{A}$ and $\mathcal{B}$ are said to be equivalent if $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$. The relation between NFA and DFA is stated in the following proposition.

Proposition 1. *(Rabin and Scott [86])*

Any language accepted by NFA is accepted by a DFA.

A non-deterministic finite automaton can be always converted to a deterministic finite automaton equivalent, that is to say which recognizes the same rational language. The methods allowing this conversion are called **powerset construction** or subset construction.They were published for the first time by Michael O. Rabin and Dana S. Scott [86] in an article published in 1959. They won the Turing Prize for their work on non-determinism in 1976. The possibility of such conversion, and the existence of an algorithm to realize it is remarkable and useful. It is remarkable because there are few machine models for which the deterministic and non-deterministic versions have the same recognition power. Deterministic Pushdown Automata are less powerful than general Pushdown Automata. It is proven also that there exist languages which are accepted by some nondeterministic Büchi-automaton but not by any deterministic Büchi-automaton. The big challenge of this determinization is that subset construction can produce an automaton whose size, measured in number of states, is exponential with respect to the size of the starting automaton. The number of state may grow exponentially when making deterministic a NFA.

2.1.2 Timed Automata

The classical models such as finite automata explained in the previous section, and many others like Petri Nets, can not express real-time systems. These systems must meet specifications, that is to say, behave as expected. In particular, they have to satisfy constraints on their response time. The correction of such systems therefore depends not only on the validity of the outputs of the system, but also on the *time* in which they are issued. The expected properties of a real time system can be represented in many ways. Depending on

the degree of criticality, knowledge of the system and properties to check, many approaches are possible.

In those systems, explicit timing constraints and quantitative properties of delays between events should be expressed. Hence the necessity of a new modeling tools able to have this expressive power. Timed automata [3, 6] are a well-established model for real-time systems. They present one of the most successful formalisms for describing the timing behaviour of computer systems. They were initially introduced by Rajeev Alur and David L. Dill in the early 1990s [3, 6]. Roughly, a timed automaton is a finite automaton equipped with continuous clocks that evolve synchronously. The arcs are labeled with guards on the clocks (e.g. the clock x is greater than 1) and can reset clocks to zero. Research around this model is very active.

2.1.2.1 Petri Nets

Examples of other formalisms with the same purpose, are timed Petri-net models [32, 107] or timed process algebras[88]. The well-known model is that of Time Petri nets [75, 17], an extension of Petri nets. Recall that Petri nets are composed of a set of places that are connected by transitions. A transition connects two sets of places and can be fired if there are enough tokens in the first set (precondition) and the activation of the transition removes the tokens specified by the pre-condition and adds tokens in the second set of places (post-condition). An execution of a Petri net starts with a given number of tokens in some place (this is called initial marking). The timed aspects can be added in different ways. In Time Petri Nets [75, 17], time constraints are put on transitions; two dates min and max are associated with each transition. When the precondition of a transition is satisfied, a clock is reset and the transition can be fired when the value of the clock belong to the interval associated with the transition. Let us suppose that the transition t became enabled for the last time at date θ, then t cannot be fired (cannot be taken) before the date $\theta + min$ and must be fired no later than date $\theta + max$, except if firing another transition disabled t before then.

Time Petri nets naturally express specifications "in delays". By making explicit the beginnings and ends of actions, they can also express specifications "in durations", their

applicability is thus broad. Note that there are other extensions of Petri nets taking into account time but they are not widely used. For example, timed Petri nets whose transitions have fixed execution times [87, 32, 107]. The temporal aspects can also concern the tokens by considering their ages as in timed arcs Petri nets [55].

2.1.2.2 Clock variable

A timed automaton is a finite automaton extended with real variables called clocks. Each clock measures the elapsed time since it was last reset. These clocks all have a slope equal to 1, i.e. they all advance at the same speed (that of universal time is often implicit in the model). Their values can be compared to constants or to each other and they can be reset. Subsequently and when the variables are clocks, we will talk about constraint of clocks to designate a constraint of variables. The time-units considered and the precision of the clocks depend on the contexts and can be very different. The behavior of a clock can be described by the diagram of Figure 2.2. In this scheme, the clock x has been reset twice: the first after 3 time units and the second after two other time units.

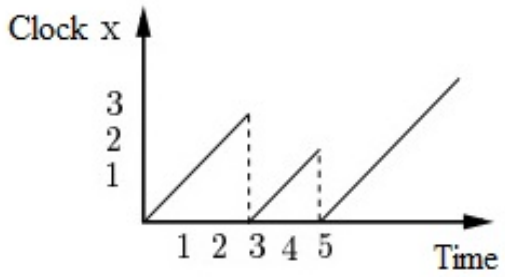

Figure 2.2: Performance of a clock

Timed automata can then be described as finite automata to which clocks have been added. There are then two types of possible evolution for such a system:

- Either the time elapses while the system remains in the same state. In this case, all the clocks evolve at the same rate by a value equal to the waiting time.

- Either it is possible to cross a transition and thus perform the action indicated on it. Beforehand, it is necessary to check that the values of the clocks satisfy the clock constraints that are placed on the transition, then, after the action is performed, the clocks specified on the transition must be reset.

Let X be a finite set of clocks. We define the set $\Phi(X)$ of clock constraints over X by the following grammar:

$\Phi ::= \mathit{true} \mid x < c \mid x \leq c \mid x > c \mid x \geq c \mid \phi_1 \wedge \phi_2 \mid \neg\phi$

where $x \in X$ is a clock, $c \in \mathbb{Q}$ is a rational, ϕ_1 and ϕ_1 are clock constraints. We note $\mathcal{C}(X)$ the set of constraints expressed over the clocks.

Definition 3. *A timed automaton is a tuple $\mathcal{A} = (\Sigma, S, S_0, F, X, I, T)$ where:*

- *Σ is a finite alphabet of events,*
- *S is a finite set of control states,*
- *$S_0 \subseteq S$ is a set of initial control states,*
- *$F \subseteq S$ is a set of accepting control states,*
- *$I : S \to \mathcal{C}(X)$ presents a finite set of state invariant,*
- *X is a finite set of clocks, and*
- *$T \subseteq S \times \Phi(X) \times \Sigma^\epsilon \times 2^X \times \Phi(X) \times S$ is the transition relation between states that presents the edges. An edge $(s, \phi, \alpha, R, \phi', s')$ allows α-labeled transition from s to s', provided the precondition ϕ on clocks is met. Afterwards, the clocks in R are reset to values satisfying the postcondition ϕ', and all other clocks remain unchanged.*

The Figure 2.3 illustrates an example of a timed automaton. A timed event is a pair (t, a), where $t \in \mathbb{R}^+$ is called the timestamp of the event $e \in \Sigma^\epsilon$. A timed word or trace is a finite or infinite sequence of timed events $w = (t_0, a_0)(t_1, a_1)(t_2, a_2)...$ whose sequence of timestamps $t_0 t_1 t_2 ...$ is non-decreasing. Its corresponding run is $\sigma = s_0 \xrightarrow{t_0} s'_0 \xrightarrow{a_1} s_1 \ldots \xrightarrow{t_{k-1}} s'_{k-1} \xrightarrow{a_k} s_k \ldots$. Since runs are infinite, some convergence propositions may appear. The most famous is the

zenoness which simply express to the convergence of the absolute time along a run, and state that it is not possible to perform an infinite number of transitions in a bounded time.

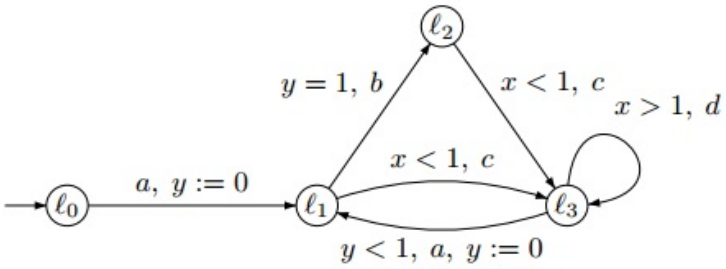

Figure 2.3: Timed automaton $\mathcal{A}$

Definition 4. *(Zenoness assumption)*

- $w = (t_0, a_0)(t_1, a_1)(t_2, a_2)...$ *is Zeno (resp. non-Zeno) if* $(t_i)_{i\in\mathbb{N}}$ *is finite (resp. infinite).*
- *An infinite run* $\sigma = s_0 \xrightarrow{t_0} s'_0 \xrightarrow{a_1} s_1 \ldots \xrightarrow{t_{k-1}} s'_{k-1} \xrightarrow{a_k} s_k \ldots$ *is Zeno (resp. non-Zeno) if the sum* $\Sigma_{i\in\mathbb{N}} t_i$ *is bounded (resp. diverges).*
- *A timed automaton is strongly non-Zeno if in every cycle* $l_1, l_2, \ldots, l_1$ *there is one clock which is reset and lower guarded by a positive constant [12].*

A timed automaton can be interpreted as a labeled transition system with an infinite set of states, $\mathcal{LTS} = (\Sigma, S, S_0, R)$ where:

- S is the set of states comprising couples (s, ν), such that s is the state and ν is the clock valuation.
- $S_0 \subset S$ is the set of initial state that consists of couples (s, ν) such that $s \in S_0$ and $\nu(x) = 0$ for every $x \in X$.
- R is the transition relation. The transition is enabled either by time achievement or by action firing.

- Delay transition: the transition is enabled after elapse of duration determined by the guard constraint. Let $d \in \mathbb{R}^+$, we note $(s, v) \xrightarrow{d} (s, \nu')$ if $\nu' = \nu + d$ and for every $0 \leq e \leq d$, $\nu + e$ fulfill the transition constraint $I(s)$.
- Action transition: The transition is enabled once the action or event which label the edge starts off. For $a \in \Sigma$, we note $(s, \nu) \xrightarrow{a} (s', \nu')$ if there is $(s, a, \phi, R, s') \in T$ such that ν satisfies ϕ and $\nu' = \nu[R := r_x]$.

Consequently, the relation R is defined as follows: $(s, \nu)R(s', \nu')$ iff there is $d \in \mathbb{R}^+$, $a \in \Sigma$ and an intermediate state (s'', ν''), such that: $(s, \nu) \xrightarrow{d} (s'', \nu'') \xrightarrow{a} (s', \nu')$. We denote $(s, \nu) \overset{a}{\Rightarrow} (s', \nu')$ iff $(s, \nu)R(s', \nu')$.

A clock valuation of $\mathcal{A}$ is a function $\nu : X \to \mathbb{R}^+$.

$$\nu[R := r_x](x) = \begin{cases} r_x & \text{if } x \in R \\ \nu(x) & \text{otherwise} \end{cases}$$

It is equivalent to say that $\nu[R := r_x]$ is what we get by reseting all variables in $R \subset X$ to rational values r_x.

In addition, if $\lambda \in \mathbb{R}^+$ then $(\nu + \lambda)(x) = \nu(x) + \lambda$ for all $x \in X$. A global state of $\mathcal{A}$ is a pair (s, ν) where $s \in S$ is a control state and ν is a clock valuation. We note: $G = S \times (\mathcal{R}_+)^X$ for the set of global state.

Automaton $\mathcal{A}$ induces an $(\mathbb{R}^+ \times \Sigma^\epsilon)$-labeled transition relation on the set of global states as follows. Write $(s, \nu) \xrightarrow{\lambda, \alpha} (t, \nu')$ iff there is an edge $(s, t, \phi, \alpha, R, \phi') \in E$ such that $\nu + \lambda$ satisfies ϕ, ν' satisfies ϕ' and $(\nu + \lambda)(x) = \nu'(x)$ for all $x \notin R$.

The initial clock valuation ν_0 assigns 0 to every clock. A run of $\mathcal{A}$ is a finite or infinite sequence of transitions

$$\pi = (s_0, \nu_0) \xrightarrow{\lambda_0, \alpha_0} (s_1, \nu_1) \xrightarrow{\lambda_1, \alpha_1} (s_2, \nu_2) \xrightarrow{\lambda_2, \alpha_2} \ldots$$

Where $s_0 \in S$ is an initial control state and $\nu_0(x) = O$ for all $x \in X$.

A finite run is accepting if the last control state is accepting. We say that an infinite run is accepting if infinitely many control states in the run are accepting.

A possible run of that automaton is:

$$(s_0,(0,0)) \xrightarrow{2.6} (s_0,(2.6,2.6)) \rightarrow (s_1,(2.6,0)) \xrightarrow{1.5} (s_1,(6,3.4)) \rightarrow (s_2,(6,3.4))$$

The notation $(s_1,(6,3.4))$ means that we are in state s_1, that the value of clock x is 6 and the value of clock y is 3.4.

Many works are dedicated to the "theoretical" understanding of timed automata: determinization [72], minimization [91], power of clocks [100], power of ϵ-transitions [14], robustness [31, 90], the logical characterizations [22], etc. Many extensions of this model are proposed [102, 73]. The "practical" aspects have also been taken into account and several tools have even been developed for timed automata model-checking, namely HyTech [57], Kronos [104] or Uppaal [20]. These model-checkers have been used to verify many real and huge studies in the industry. It is worth to mention that many algorithmic aspects have been considered and thoroughly conducted.

One of the strengths of timed automata is that reachability properties are decidable in polynomial space [10], though timed automata have an infinite number of configurations. The core of this result is the construction of the so-called region automaton, which finitely abstract behaviours of timed automata in such a way that checking reachability in a timed automaton is reduced to checking reachability in a (somewhat larger) finite automaton. This construction has many other applications, as for example the decidability of the TCTL model-checking [29] (TCTL is the timed extension of the logic CTL). However, many problems remain undecidable, as not everything can be reduced to the untimed framework. Moreover, this low complexity can not be achieved in other formalisms, it is undecidable for temporal Petri nets [64] and TC-MSC graphs [52]. Nevertheless, an abstraction has been developed to solve the problem of accessibility in temporal Petri nets bounded in polynomial space [23]. Unfortunately, the limited character of temporal Petri nets is undecidable.

Theorem 1. *Reachability (or equivalently emptiness) is decidable for timed automata. It is a PSPACE-complete problem (for both diagonal-free as well as general timed automata).*

The problem with timed automata is that the number of configurations of a timed automaton is infinite (a configuration is a pair (q, ν) where q is a state and ν a clock valuation). This is because clocks in a timed automaton are real-valued, the state-space is then infinite

and not a good base for automatic verification. Thus, classical techniques used for verifying finite automata can not be used for timed automata. In other words, the main difficulty of the model-checking of real-time systems defined as timed automata is that uncountably many states have to be analyzed, since the semantics of the time automata are defined by an infinite transition system. Specific symbolic techniques and abstractions have to be developed, and should take into account the specific properties of timed automata, in particular the fact that clocks evolve synchronously with global time.

2.2 Model-checking

Model-checking is an automatic procedure that allows to test algorithmically whether a given model, the system itself or an abstraction of the system, satisfies a logical specification, usually formulated in terms of temporal logic. The figure 2.2 depicts the general scheme of the model-checking technique. Suppose that we are given a system (a specification for example) and a property of this system. The preliminary stage of model-checking consists, on the one hand, in modeling the system, that is, in constructing an object in the well-defined formal framework that "reproduces" as accurately as possible the behavior of the real system, and on the other hand to model the property to be verified in a suitable specification language. Model-checking then consists in ensuring that the model constructed for the system verifies the property expressed in the aforementioned language (hence the term "model-checking", these are the models that we check). To achieve this, it is of course necessary to have model-checking algorithms.

If $\mathcal{A}$ is a timed automaton representing the system, and $\mathcal{P}$ a timed automaton representing the property, verifying that $\mathcal{A}$ satisfies the property $\mathcal{P}$ corresponds to checking that all behaviours of $\mathcal{P}$ are also behaviours of $\mathcal{A}$. This is an inclusion question.

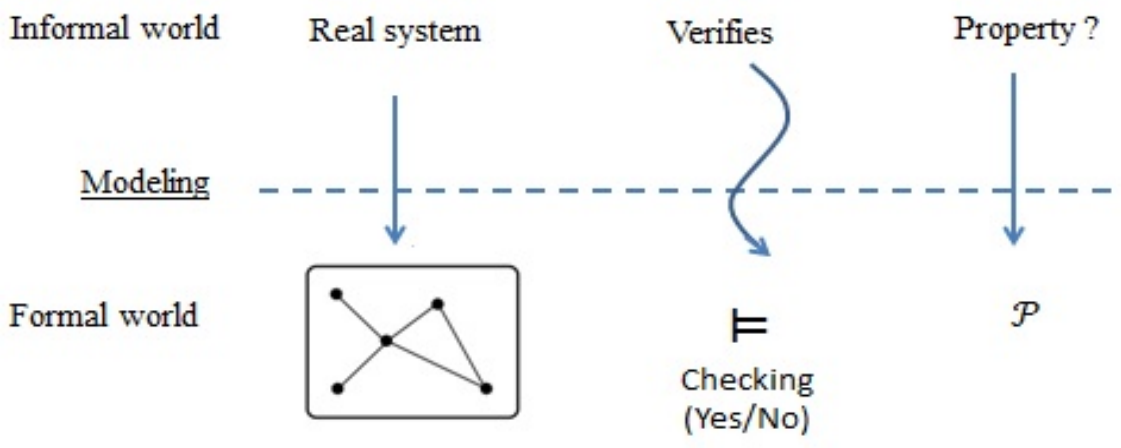

Figure 2.4: General scheme of model-checking

2.2.1 Temporal logic

When verifying a system, the first step after modeling the system is to express the properties we aim to verify on the dynamic behavior of the system. The complexity and difficulty of specification is related to the nature of system. Real-time systems need to be able to specify properties with time intervals defined by time variables, clocks. In simpler cases such as those we are going to study, the properties simply refer to the scheduling of events over time. In this case, temporal logics are used.

The temporal logic was introduced by Arthur Prior in 1957 to process temporal information in a logical framework close to the modal logic. This later consists in the study of possible inferences from expressions like "it is necessary that..." Or "it is possible that...". It also includes other systems of deduction, such as temporal logic ("a day in the future ...", "always in the future ...", ...) or deontic logic ("it is obligatory that ... "," it is authorized that ... ", ...).

Indeed, timed temporal logics in model-checking extends classical untimed temporal logics with timing constraints. For instance, we can write a formula like $\mathbf{G}$ (Train $\Longrightarrow$ $\leq$**10min** Close) to express the following quantitative property: every time the train passes, the barrier must be closed within 10 minutes. This kind of properties cannot be expressed using untimed or standard temporal logics, as those logics can only speak about the relative order of events, not about the quantitative distance (in time) between these events.

2.2.1.1 Linear Temporal Logic

Linear Temporal Logic (LTL) allows us to express properties on state sequences, so-called path formulas, which makes it more natural and easier to understand than other temporal logic such as CTL explained in the following paragraphs. However, the verification of LTL formulas is much more expensive [99], and requires an exponential time (it is PSPACE-complete). The counterpart to the fact that the verification of LTL formulas is difficult is that the search is more active, with the aim of finding ever more efficient model-checking algorithms.

Definition 5. *Let* $\mathbf{AP}$ *be a non empty set of atomic propositions (representing the events and states of the system) and let* $\Sigma = 2^{\mathbf{AP}}$. *We note* Σ^ω *the set of infinite words on the alphabet* Σ. *The syntax of the Linear Temporal logic is defined as follows:*

$$\varphi ::= p|\neg\varphi|\varphi \vee \psi|\mathbf{X}|\varphi\mathbf{U}\psi \tag{2.1}$$

Where p is an element of $\mathbf{AP}$ *and* $\mathbf{X}$ *and* $\mathbf{U}$ *are called modalities:* $\mathbf{X}\varphi$ *is read "next* φ*" and means that* φ *will be true in the next state,* $\varphi\mathbf{U}\psi$ *is read "*φ *until* ψ*" and means that* φ *holds true until that* ψ *will be true. These modalities concern the future, while* $\mathbf{X}^{-1}$*(means previous) and* $\mathbf{S}$ *(means since) are modalities of the past. The Past Linear Temporal Logic, noted PLTL, is defined with the same syntax including in addition these two later modalities.*

The syntax proposed in this definition is the usual syntax of LTL, but in the practice other operators and modalities are defined and deduced from this syntax: for example, we authorize the boolean operator $\wedge$ which is deduced from the operator $\vee$ as follows: $\varphi \vee \psi \equiv \neg(\neg\psi \wedge \neg\varphi)$. In the same line, we use the propositions, which designate respectively always true and always false, and they are defined in this way: $\top \equiv p \vee \neg p \qquad \bot \equiv \neg\top$.

We can also define new modalities such as:

$$\mathbf{F}\varphi \equiv \top\mathbf{U}\varphi \qquad \mathbf{G}\varphi \equiv \neg\mathbf{F}\neg\varphi$$

Intuitively, $\mathbf{F}\varphi$ designates that φ will be satisfied at a point in the future, it is read "eventually φ". While $\neg\mathbf{F}\neg\varphi$ means that it is not true that the negation of φ will be

satisfied in the future, i.e. that φ holds true always in the future. The modality $\mathbf{G}\varphi$ is read "*always* φ".

The formulas of linear time logic express properties on an run of a (model of a) reactive system. This run is represented by the sequence of states that the system goes through over time. Such a sequence is called "path".

As said, this logic describes properties of individual executions, and its semantics are defined as a set of executions. Now, We can define its semantic which is valid for both finite and infinite paths.

Definition 6. *Let π be a path, finite or infinite, i a natural integer strictly inferior to the path length, φ and ψ two formulas of LTL, and p an atomic proposition of* **AP**. *We define the relation $\models$ inductively, from the following rules:*

$\pi, i \models \neg\varphi$	iff	$\pi, i \models \varphi$
$\pi, i \models \varphi \wedge \psi$	iff	$\pi, i \models \varphi$ or $\pi, i \models \psi$
$\pi \models \mathbf{X}\varphi$	iff	$i + 1 < leng(\pi) \wedge \pi, i + 1 \models \varphi$
$\pi, i \models \varphi\mathbf{U}\psi$	iff	$\exists k.((i \leq k \leq leng(\pi)) \wedge (\pi, k \models \psi) \wedge (\forall i \leq j < k.\pi, j \models \varphi))$
$\pi, i \models p$	iff	$\pi(p) \models \top, \forall p \in \mathbf{AP}$

Table 2.1: Satisfaction relation for LTL

leng(π) denotes the length of the path π, it represents the cardinal of set of states belonging to this path. When $\pi, i \models \varphi$ is true, we say that the formula φ is true, or is satisfied, in the position i along the path π.

Intuitively, asserting that two LTL formulas are equivalents indicates that they have the same "meaning" and they are satisfied simultaneously, for the same paths and same positions.

Definition 7. *Let φ and ψ be two LTL formulas. Let Π be a set of paths. The formulas φ and ψ are said globally equivalent (or, more simply, equivalent) for Π when the following property is verified:*

$$\forall \pi \in \Pi, \forall i \in \mathbb{N}, (\pi, i \models \varphi \Leftrightarrow \pi, i \models \psi) \tag{2.2}$$

The expressiveness is the great stake of temporal logics. It is shown that there is a linear time translation of LTL to the First-Order Modal LOgic (FOMLO) [50], since the LTL

semantics are defined based on FOMLO formulas. A property of FOMLO is a logic formula constructed from a set of variables, existential and universal quantifiers on these variables, Boolean operators and predicates and function defined above, as well as the modal operators for possibility and necessity. These formulas are interpreted on $\mathbb{N}$ in this context, i.e. on paths. We can define several restrictions of the FOMLO logic: we note $\mathbf{FO}^n$ a FOMLO fragment wherein one can use only n variables.

The fact that LTL is as expressive as FOMLO also gives us the following result: LTL defines exactly the "star-free" languages. Each LTL formula can therefore be associated with a counter-free Büchi automaton.

Decidability of LTL property verification

The most important step in the development of temporal logics is the study of their costs, in terms of the complexity of the verification problems. First, we will define the verification problems that we consider, then we will study the complexity of these problems for the different fragments that we have defined previously.

Satisfiability and validity checking

The problem of validity is expressed as follows: "Given a formula φ, is it true along any path?". More formally:

Definition 8. *Let φ be a LTL formula. We say that φ is globally valid if, for all paths π and positions i, $\pi, i \models \varphi$, and we assert that φ is initially valid if, for all paths π, $\pi \models \varphi$.*

Intuitively, asserting that "φ is valid" means that φ is true anywhere, along the considered paths. Saying that "φ is initially valid" means that φ is true in the initial state for all paths (similar to say that this formula is equivalent to $\top$).

The problem of satisfiability is the "existential" version of validity: a formula is satisfiable if there is a path that satisfies it.

Definition 9. *Let φ be a LTL formula. This formula is said to be globally satisfiable if there*

exists a path π and position i such that $\pi, i \models \varphi$. In the same, φ is initially satisfiable if there exists a path π such that $\pi \models \varphi$.

2.2.1.2 Branching-time temporal logic

The branching-time temporal logics are non-linear temporal logics. Computation Tree Logic (CTL) belongs to the family of branching-time logics, meaning that its model of time is a tree-like structure in which the future is not determined; there are different paths in the future, any one of which might be the actual path that is followed.

CTL describes properties of a computation tree: formulas can reason about many executions at once. It uses the temporal operators **X**, **F**, **G** and **U** of LTL, and defines two others quantifiers; **A** and **E** that express "all paths" and "exists a path", respectively. The syntax of the CTL is as follows:

$$\varphi ::= a \mid \neg\varphi \mid \varphi \wedge \psi \mid \mathbf{E}\varphi\mathbf{U}\psi \mid \mathbf{A}\varphi\mathbf{U}\psi \tag{2.3}$$

$\mathbf{X}\varphi$ (said *next*φ) is a temporal formula designating that the formula φ is verified in the state that follows the current state. $\varphi\mathbf{U}\psi$ (said φ *until* ψ) means that φ is true in all states achieved from the current state and residing before the state from which the formula ψ becomes true. These formulas are interpreted on a given path. Two quantifiers allows to express different paths:

- $\mathbf{A}\varphi$ (*for all paths* φ) means that the formula φ is true along all paths issued from the current states.
- $\mathbf{E}\varphi$ (*for some paths* φ) means that the formula φ is true along at least one path issued from the current states.

2.2.1.3 Timed temporal logics

TCTL logic (TCTL stands for Timed Computation Tree Logic) is a timed extension of the classical untimed branching temporal logic CTL [40], with time constraints on modalities. It has been defined in [3]. Slightly different syntaxes are proposed, for example in [56, 103, 98].

It is a propositional tree logic that applies on timed transition systems, so in particular on timed automata. We use the presentation of TCTL made in [98]. The other presentations are of course equivalent.

If I is the set of integer or infinite bound intervals of $\mathbb{R}^+$ and if $\mathcal{AP}$ is a set of atomic propositions, the TCTL formulas built on $\mathcal{AP}$ are defined inductively by the following grammar:

$$\varphi ::= a \quad | \quad \neg\varphi \quad | \quad \varphi \wedge \psi \quad | \quad \mathbf{E}\varphi\mathbf{U}_I\psi \quad | \quad \mathbf{A}\varphi\mathbf{U}_I\psi \tag{2.4}$$

Where $a \in \mathcal{AP}$ is an atomic proposition. The logic combinators are those of the propositional logic: negation ($\neg\varphi$), conjunction ($\varphi \wedge \psi$) and disjunction ($\varphi \vee \psi$).

The TCTL formulas are interpreted on the states of the labeled timed transition systems. In fact, there are two possible semantics for TCTL, the first is said 'continuous', and the second is more discrete and is said 'pointwise'. These two semantics have the same satisfaction relationship as defined in the table 2.2, and only differ in the interpretation of the term 'position'. In the continuous semantics, a position in a run σ is any state appearing along σ. For instance, if there is a transition $(s, \nu) \xrightarrow{\gamma,e} (s', \nu')$, then any state $(s, \nu + t)$ such that: $0 \leq t \leq \tau$ is a position of σ, and obviously (s', ν'). In the pointwise semantics, the positions in this transition is just the states s and s'.

As said, the models of temporal formula are automata with properties on states, without vocabulary, such that the transition function T is an application from the state set Q to $\mathcal{AP}(Q)$, called *structures de Kripke*. The following table shows the rules for basic modalities and the satisfaction relationship for TCTL (i.e. the transition function).

$s \models \mathbf{tt}$	if	always true
$s \models p$	if	s is labled with p (i.e. $p \in \gamma(s)$)
$s \models \neg\varphi$	if	$s \not\models \varphi$
$s \models \varphi_1 \vee \varphi_2$	if	$s \models \varphi_1$ ou $s \models \varphi_2$
$s \models \mathbf{E}\varphi_1\mathbf{U}_I\varphi_2$	if	there exists a path $s \rightsquigarrow s'$ such that $s' \models \varphi_2$, the path time is in I and φ_1 holds along this path.
$s \models \mathbf{A}\varphi_1\mathbf{U}_I\varphi_2$	if	on any path starting from s, there exists $s \rightsquigarrow s'$ such that $s \models \varphi_2$, the path time is in I and φ_1 holds along this path.

Table 2.2: Satisfaction relation for TCTL

The temporal combinators allows to construct expressions that should be verified on certain states along a given run or along of given paths.

The above language is often enhanced with new constructions useful in practice. Intuitively, a formula $\mathbf{E}\varphi_1\mathbf{U}_I\varphi_2$ expresses that from the current configuration, it is possible to follow a path whose prefix, of duration in the interval I, verifies φ_1 at each instant and the last configuration of this prefix verifies φ_2. The formula $\mathbf{A}\varphi_1\mathbf{U}_I\varphi_2$ expresses that the previous property holds for any path starting from the current configuration. $\mathbf{tt} \equiv a \vee \neg a$ stands for true and $\mathbf{ff}$ for false. We define also:

- The eventuality operator $\mathbf{F}_I\varphi \equiv tt\mathbf{U}_I\varphi$ (read *eventually* φ), meaning that the formula φ is true in at least one state achieved in the future from the current state in the current run. It is noted $\Diamond\varphi$.
- The globally operator $\mathbf{G}_I\varphi \equiv \neg(\mathbf{F}_I\neg\varphi)$ (read *henceforth* φ) means that the formula φ is true in all future states achieved from the current state in the current run, it is noted $\Box\varphi$.
- The weak until operator $\varphi W\psi$ (read *until possibly* φ) means that the formula φ is true in all states achieved from the current state and residing before the state if its exists from which the formula ψ becomes true.

The logic is also enriched with new modalities expressing the past formula. The following notations are very often used:

$$\mathbf{EF}_I\varphi \equiv \mathbf{E}(tt\mathbf{U}_I\varphi) \qquad \mathbf{AF}_I\varphi \equiv \mathbf{A}(tt\mathbf{U}_I\varphi)$$
$$\mathbf{EG}_I\varphi \equiv \neg(\mathbf{AF}_I(\neg\varphi)) \qquad \mathbf{AG}_I\varphi \equiv \neg(\mathbf{EF}_I(\neg\varphi))$$
$$\varphi_1 \implies \varphi_2 \equiv (\neg\varphi_2) \vee \varphi_2$$

For instance, $\mathbf{EF}_{[0,5]}\varphi$ expresses that it is possible to achieve, in less than 5 units of time, a configuration that verifies φ, while $\mathbf{AG}\varphi$ means that any configuration that can be reached satisfies φ.

TCTL is given with external clock variables. That is, we can use variables to express timing constraints. Let us take this formula as example: $\mathbf{AG}(P \implies x.\mathbf{AF}(P' \wedge x \leq 14))$.

We interpret this property as follows: each time a P is encountered, a clock x is reset, and we check that along all possible runs, later, P' holds and the value of the clock x is not more than 14. Compared to the TCTL with intervals constraining the modalities, explained above, it has been proved in [30] that TCTL with external clock variables is strictly more expressive.

Decidability

In TCTL , the time is "built-in", in the inverse of the first order logic. We manipulate it with a reduct number of chosen combinators, and actions occur in multiple discrete instants $0, 1, 2, ..., n, ...$ of elementary time. This remark helps to study the decidability of the TCTL model-checking problem.

The decidability and complexity of the model-checking problem for TCTL was being studied, and the following theorem is proved, under the continuous and the pointwise semantics, since the beginning:

Theorem 2. *The model-checking problem for TCTL is PSPACE-Complete [3, 4].*

The proof of this result relies on the construction of a region automaton, an extension of the classical region automaton construction for deciding language emptiness in timed automata.

2.2.1.4 Probabilistic temporal logic

In the case of probabilistic model-checking, we use models that incorporate information about the probability that a transition between states will occur. There are various models used to describe the behavior of a probabilistic system. We can cite the Discrete-Time Markov Chain (DTMC) [34] which are the simplest model representing probabilistic systems. These are transition systems where transitions are labeled by the probability of moving from one state to another.

PCTL (Probabilistic Computational Tree Logic) introduced by H. Hansson et B. Jonsson [19] is an extension of the temporal logic CTL.

Syntax of PCTL

Let AP be a finite set of atomic proposition, the syntax of PCTL is defined as follows:

$$\varphi ::= \mathbf{tt} \mid a \mid \neg\varphi \mid \varphi \wedge \psi \mid P_{\sim p}(\varphi) \mid \mathcal{X}\varphi \mid \varphi_1 \mathcal{U} \varphi_2 \mid \varphi_1 \mathcal{U}_{\leq k} \varphi_2$$

where $a \in AP$, $\sim \in \{<, \leq, , \geq, >\}$ and $p \in [0, 1]$.

The first five formula are called state formula and the last three are path formula. This presentation is called the Backus-Naur presentation [6]. It means that formulas of the PCTL logic are built recursively from this syntax. The operator which concerns the probability is $P_{\sim p}(\varphi)$. Intuitively, a state satisfies the formula $P_{\sim p}(\varphi)$ if the probability of taking a path from a state s that satisfies φ is within the range defined by $\sim p$.

Concerning the semantic, it is the same as CTL with this addition:

$$s \models P_{\sim p}(\varphi) \text{ iff } Prob(\{\pi \in Paths(s) | \pi \models \varphi\}) \sim p$$

This equivalence means that is satisfied in a state s, if and only if there is a path π from the state s such that $Prob(\{\pi \in Paths(s) | \pi \models \varphi\})$ is within the range defined by $\sim p$. For example $P_{\geq p}(\mathcal{X}\phi)$ is satisfied by the model M if the probability of the path set starting from the initial state s_0 and the next state s_1 satisfies φ is great or equal to p.

2.2.1.5 Temporal properties

Safety: "something bad will not happen"

liveness: "Something good will happen".

Fairness: Every process should be executed infinitely often: if we attempt/request infinitely often, then we will be successful/allocated infinitely often.

It is really useful when scheduling processes, responding to messages, etc.

Deadlock: Is the case where the system is blocked in a state without being able to progress. It is unable to change its state indefinitely and the length of stay in this state is infinite.

2.2.1.6 Model-checking algorithm

The overall structure of the model-checking algorithm of PCTL and TCTL derives from the CTL model-checking algorithm presented in [15]. The algorithm consists of:

- Constructing the syntax tree of the formula ϕ.
- Cross recursively the syntax tree from bottom to top (from the leaves to the root), determining the set of $Sat(\phi) = \{s \in S | s \models \phi\}$, for each sub-formula $\varphi \in \phi$.

The syntax tree of a formula ϕ is a tree whose every node is a sub-formula of ϕ and the root of the tree is the formula ϕ itself. The leaves of the tree are atomic propositions or the Boolean operator true for the case of PCTL. An example of this tree is depicted on figure 2.5.

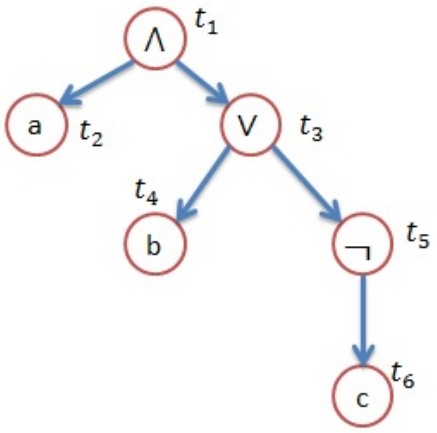

Figure 2.5: Syntax tree of $\phi = a \wedge (b \vee \neg c)$

2.2.2 Symbolic Analysis of Timed Automata

Several problems of timed automata model (such as the problems of the determinizability or the universality of timed languages recognized by timed automata) are undecidable, but the main reason for the success of this model is the abstraction of regions which allows, for example, to decide the problem of state reachability in polynomial space [6]. More generally,

the automaton of the regions makes it possible to decide the safety properties, the regular properties or the non-timed properties expressed in linear temporal logic (LTL) or in tree logic (computation tree logic or CTL).

A naive graph analysis in the usual state graph is therefore not feasible. Instead, the basic idea is to consider a finite quotient of this transition system, the so-called region transition system. The states in the region transition system are equivalence classes of states in the transition system that all satisfy the same atomic clock constraints, and from which "similar" time-divergent paths emanate. As the number of equivalence classes is finite, this provides a basis for model checking.

In practice, the equivalence classes are calculated "on-the-fly", during symbolic reachability analysis. Symbolic reachability analysis is a powerful paradigm for verification of infinite state systems. Symbolic reachability analysis uses finite structures to represent infinite sets of configurations, and iterative exploration procedures to compute the set of all reachable configurations, or an upper approximation of this set. This technique is used for verification of infinite systems like timed systems.

2.2.2.1 Region abstraction

The symbolic analysis is based on the construction of the region automaton. This actually consists in a finite graph simulating the timed automaton based on a finite index equivalence relation defined on the clock valuations. It is a question of identifying two valuations ν and ν' such that the behavior of the timed automaton remains the same in the configurations (q, ν) and (q, ν') for all states q in the automaton. The quotient graph of this relation of equivalence is called the graph of the regions. It is simply a finite partition of the set of possible clock valuations, for which each element is called "region". The idea of the abstraction of regions is that some clock valuations are similar and can be processed together, for example $(x = 2 : 3, y = 5)$ and $(x = 2 : 35, y = 5)$. More precisely, regions are equivalence classes on clock valuations. Two valuations belong to the same region if they satisfy the same guards and if their successors will satisfy the same guards. That is to say, from a state with one or the other of the valuations, one can follow the same paths in the timed automaton. In other words, these partitions should respect two properties:

- Compatibility between regions and constraints. Two valuations in the same region must satisfy the same constraints.
- Compatibility between regions and the passage of time. All valuations of the same region must have the same immediate successor, i.e. by letting the time flow, the valuations must first reach the same region.

This transformation has good outcomes. In fact, the size of the region automaton thus defined is exponential in the size of the timed automaton. As the reachability in the finite automata is a problem Nlog-SPACE, one obtains that the problem of reachability in timed automata is PSPACE. It's even a PSPACE-complete problem.

2.2.2.2 Zone

The set of configurations of a timed automaton is infinite. To verify this model, it is therefore necessary to be able to manipulate sets of configurations, and thus to have a symbolic representation. The most commonly used is the *symbolic representation* called zone. A zone of timed automaton $\mathcal{A} = (\Sigma, S, S_0, F, X, I, T)$ is a pair (l, Z) where $l \in S$ is the system location and Z is a polyhedron containing possible valuations of system's clocks. In other words, a zone is a set of valuations defined by clock constraints: the zone associated to the constraint C is $\{\nu \in \mathbb{T}^X | \nu \models C\}$.

Example 1. *The zone associated to the constraint* $C = x > 1 \wedge y \leq 4 \wedge x - y < 3$ *is illustrated in the figure 2.6.*

For two zones $H = (l, Z)$ and $H' = (l', Z')$, the operations of inclusion and intersection are defined in the following way:

$$H \subseteq H' \Leftrightarrow l = l' \wedge Z \subseteq Z' \tag{2.5}$$

$$H \cap H' = \begin{cases} (l, Z \cap Z') & if\ (l = l') \wedge (Z \cap Z' \neq \emptyset) \\ \emptyset & otherwise \end{cases} \tag{2.6}$$

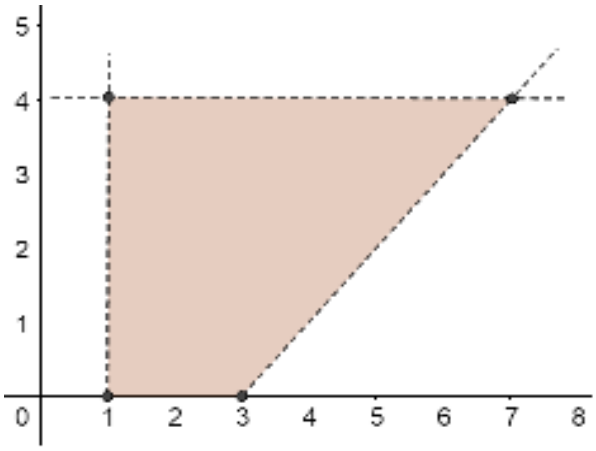

Figure 2.6: Example of a zone

The region graph construction presented so far is still expensive, other methods are used in addition to decide the problem of reachability. These are based on on-the-fly construction techniques to avoid building all configurations at the beginning. In addition, the sets of configurations will be represented symbolically.

On-the-fly analysis There are mainly two techniques for on the fly reachability algorithms: forward analysis, and backward analysis. The forward (resp. backward) analysis consists in computing all the successors (resp. predecessors) of the initial (resp. final) configurations. Recall that (q, ν) is an initial configuration if $q \in I$ and $\nu(x) = 0$ for all $x \in X$ and final if $q \in F$.

Let $H(l, Z)$ be a zone and $t = (l, Z, a, r, l')$ be a transition of $\mathcal{A} = (\Sigma, S, S_0, F, X, I, T)$.

Forward analysis with the operation *post()*

The operation *post(H,t)* returns a zone that contains all states that can be reached by $\mathcal{A}$ after it performs transition t from zone H. The operation *post(H,t)* is defined as follows:

$$post(H,t) = (l', ((Z^{\uparrow} \cap g)[r := 0]) \cap Inv(l')) \tag{2.7}$$

Example 2. *Let us consider that the automaton $\mathcal{A}$ depicted in the Figure 2.7 is in a zone (l, Z). The polyhedron $Z^{\uparrow}$ illustrates all valuations that can be reached from Z by time elapse. In addition, $\mathcal{A}$ can stay in the location l only when the valuations of its clocks satisfy the*

invariant Inv(l). A transition t may be performed only if the clocks' valuations belongs to the polyhedron Z_t; this means that at the moment of executing t possible valuations are defined by intersection of all valuations that could be reached from Z in the location l with the guard Z_t: $(Z^{\uparrow} \cap Inv(l))$. At the moment of executing t the set of clocks r is reset, what is symbolically represented by performing forward clock reset operation. Since the invariant for l' becomes satisfied, the zone reached directly after performing t is defined by $(Z^{\uparrow} \cap Inv(l))[r := 0]$.

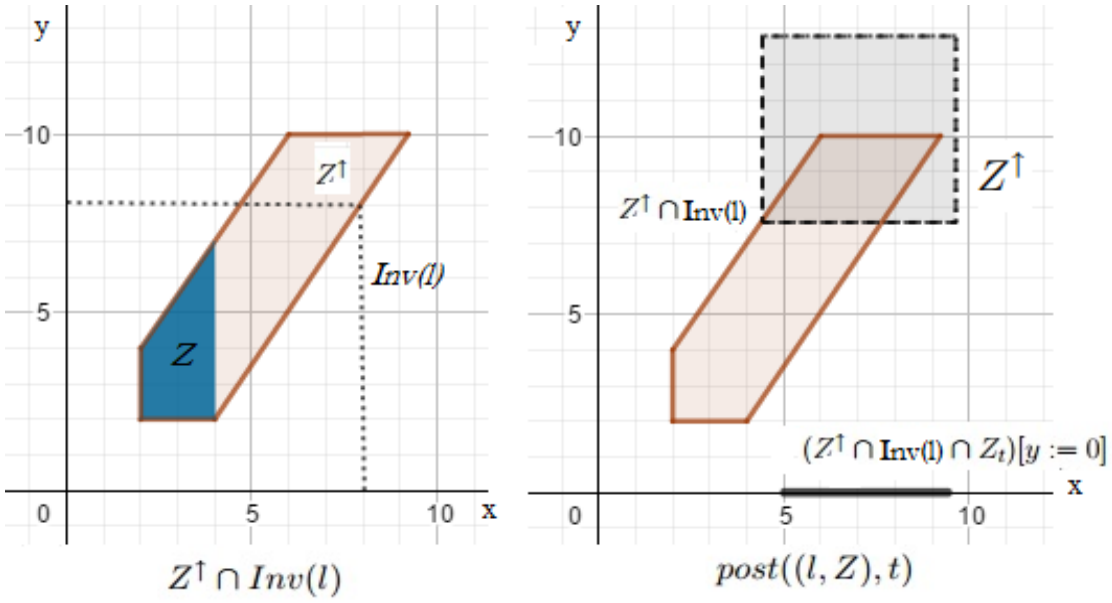

Figure 2.7: The *post* operation

Backward analysis using the operation *pred()*

The operation *pred()* is a reverse operation to *post()*. Let us assume that the automaton $\mathcal{A}$ is in the zone $H = (l', Z')$ and suppose a transition $t = (l, Z_t, a, r, l')$ ending in the location l'. Then, the operation *pred()* returns a zone that can be occupied before performing the transition t such that execution of t results in a state in H. The operation *pred(H,t)* is defined as follows:

$$pred(H,t) = ([r := 0]Z \cap Z_t \cap Inv(l))^{\downarrow} \cap Inv(l) \tag{2.8}$$

post-pred stability. Let $H = (l, Z)$, H_1 and H_2 three symbolic states and $t_1 = (l_1, Z_1, a_1, r_1, l)$ and $t_2 = (l, Z_2, a_2, r_2, l_2)$ be two transitions of TA $\mathcal{A}$. Assume that $H = ((post(H_1, t_1)))$. Then, post-pred stability means that for all states (l, ν) there must be a state in H_1 that allows reaching $(l, \nu) \in H$ by performing transition t_1. Also, for all states $(l, \nu) \in H$ it is possible to reach a state in H_2 by performing transition t_2.

- *pred stability for the post operation*: For all $q = (l, \nu) \in H$, there exist $q_1 = (l_1, \nu_1) \in H_1$, and $d_1 \geq 0$ such that $\nu_1 + d_1 \in Z_1$ and $\nu = (\nu_1 + d_1)[r_1 := 0]$. We say that H is *pred-stable* to H_1 by t_1. q_1 is called predecessor state of q.

- *post stability for the pred operation*: For all $q = (l, \nu) \in H$, there exist $q_2 = (l_2, \nu_2) \in H_2$, and $d_2 \geq 0$ such that $\nu_2 + d_2 \in Z_2$ and $\nu = (\nu + d_2)[r_2 := 0] = \nu_2$. We say that H is *post-stable* to H_2 by t_2. q_2 is called successor state of q.

Symbolic path analysis. From these defined operations we can analyze the whole region graph. Let $\pi = [t_1, ..., t_n]$ be a sequence of transitions of an automaton $\mathcal{A}$ such that $t_i = (l_{i-1}, Z_i, a_i, r_i, l_i)$, for all $i \in [1, n]$ and let $\mathcal{C}$ be a set of clocks of $\mathcal{A}$.

The **forward path analysis** is carried out using the $post()$ operation. It is expressed as follows: For all $i \in [0, n]$,

$$\begin{cases} H_i = (l_0, \mathbf{zero}), & if\ i = 0 \\ H_i = post(H_{i-1}, t_i), & otherwise \end{cases} \tag{2.9}$$

Intuitively, the initial state of the system H_0 is the initial location with all clocks x are set to 0 (**zero** denotes the polyhedron constrained by $\bigwedge_{x \in \mathcal{C}} x = 0$. Then next zones H_i are obtained by $post()$ operation and are pred-stable to H_{i-1} by $t - i$.

From an algorithmic point of view, we can now describe more precisely how this analysis works. It simply consists, starting from the initial state q_0 of the automaton and from the initial zone Z_0, in computing the successors in one step, using the *post()* operator. Then we iterate this operation until the stabilization or discovery of a final state.

Lemma 1. *The final location l_n of the path π is reachable if and only if the zone H_n is not empty.*

The symbolic path $S^+(\pi)$ associated with path π is a sequence of zones obtained by forward analysis of π.

$$S^+(\pi) : H_0 \xrightarrow{t_1} H_1 ... H_{n-1} \xrightarrow{t_n} H_n \tag{2.10}$$

The **backward path analysis** is carried out using the *pred*() operations. It is then expressed formally as follows: For all $i \in [0, n]$,

$$\begin{cases} H_i = (l_n, Inv(l_n)), & if \quad i = n \\ H_i = pred(H_{i+1}, t_{i+1}), & otherwise \end{cases} \tag{2.11}$$

With the same logic, the initial state of the system H_n is the final location respecting the invariant $Inv(l_n)$. Then, next zones H_i are obtained by the *pred*() operation and are post-stable to H_{i+1} by t_{i+1}.

Lemma 2. *The final location l_n of the path π is reachable if and only if the zone $H_0 = (l_0, \mathbf{zero})$ is not empty.*

The symbolic path $S^-(\pi)$ associated with path π is a sequence of zones obtained by backward analysis of π.

$$S^-(\pi) : H_0 \xrightarrow{t_1} H_1 ... H_{n-1} \xrightarrow{t_n} H_n \tag{2.12}$$

Forward-backward analysis The pred-stability property of the *post* guarantee that each state $q \in H_i$ has a predecessor in H_{i-1}. On the other hand it does not guarantee that from all $q \in H_i$ the successor H_{i+1} can be reached. The latter is guaranteed by the post-stability property of the *pred*() operation. Therefore operations *post*() and *pred*() can be combined in order to formulate forward-backward analysis of the path π.

For all $i \in [0, n]$:

$$\begin{cases} H_i = (l_0, \mathbf{zero}) & if \ i = 0 \\ H_i = post(H_{i-1}, t_i) & otherwise \end{cases} \tag{2.13}$$

and

$$\begin{cases} H'_i = H_i, & if\ i = n \\ H'_i = H_i \cap pred(H'_{i+1}, t_{i+1}), & otherwise \end{cases} \tag{2.14}$$

By this, each H'_i verifies the post/pred stability property for all $i \in [0, n]$.

Finally, a symbolic path $S^+_-(\pi)$ is defined as sequence of zones obtained by forward-backward analysis of π.

$$S^+_-(\pi) : H'_0 \xrightarrow{t_1} H'_1 ... H'_{n-1} \xrightarrow{t_n} H'_n$$

However, many verification problems such as language emptiness, reachability analysis and model-checking of timed logics can still be decided by constructing a region graph based on the automaton under consideration and perform the analysis on this graph [2].

Before constructing the region graph of an automaton, it is mandatory to get its reduced form. An automaton, whether deterministic or not, may possess superfluous states, in the sense that they can simply be removed without changing the recognized language.

2.2.2.3 Complexity results over timed automata analysis

Closing properties of timed automata: The languages accepted by timed automata satisfy the following closure properties:

Proposition 2. *The set of languages accepted by timed automata is closed by union and intersection. Contrariwise, this set is not closed by passing to the complementary [3, 8].*

The class of timed languages accepted by a deterministic timed automaton is closed also by passing to the complement. Unfortunately, this class is much less expressive than the class of classical timed automata.

These properties are very important and have impact on the complexity of the major timed automata problems.

Emptiness problem

Lemma 3. *A language of an automaton (A) is non-empty if and only if one at least of states of F is accessible.*

The emptiness, the finitude and the infinitude of the language recognized by an automaton are decidable in linear time. Let us note that the problem of the fullness of the language recognized by a deterministic finite automaton (to know if it recognizes exactly Σ^*) has the same complexity as the problem of the emptiness, since they are inter-reducible by exchange of accepting and non-accepting states. This is not true for non-deterministic automata.

Theorem 3. *The emptiness problem on timed Büchi automaton is PSPACE-Complete.*

Universal problem Given a timed automaton $\mathcal{A}$, this problem consists in testing if $L((A)) = (\Sigma \times \mathbb{T})$. The non closing by passing to the complement of the languages recognized by timed automata makes that the universal problem can not be reduced to the emptiness problem. It is then of a different complexity:

Theorem 4. *The universal problem is undecidable for timed automata [8].*

However, for some timed automata classes such as the case with only one clock, the universality problem, have recently been shown to be decidable by J. Ouaknine and J. Worrell [80]. The same result is shown for the inclusion problem since those two problems are equivalent. The problem becomes undecidable once the ϵ transitions are allowed.

Theorem 5. *The universality problem for the class of timed automata with one clock and ϵ-transitions is undecidable.*

Reachability problem: The most fundamental properties that one should be able to verify are reachability properties. Safety properties can for example be expressed as reachability properties. Usually a class of models is said decidable whenever checking reachability properties in this class is decidable. Otherwise this class is said undecidable. For timed automata, the control-state reachability problem asks, given an automaton $\mathcal{A}$ and a control state $t \in S$, whether there is a computation $\mathcal{C}$ starting in global state (s_0, ϵ) and ending in global state (t, ϵ). This problem is equivalent to the emptiness problem (from a language-theoretical point of view), where the question is whether the language accepted by a timed automaton is

empty or not. Note that real-time systems can also be modeled with great expressiveness by hybrid automata that generalize timed automata by replacing clocks by continuous functions whose values are written by differential equations [5]. Unfortunately, expressiveness has a cost and classes of hybrid automata for which the problem of accessibility is decidable are very restrictive

2.2.3 Parametric real-time reasoning

Traditional approaches of real-time systems verification are limited to checking their correctness with respect to concrete timing properties (i.e. time constraints are defined with concrete values (reals or integers), for example: an acknowledgment will be sent within 3 seconds after receiving the message). Therefore, one has to provide the values of all timing bounds that occur in the system. But in the general context, the system should be designed relative to certain parameters of the environment which makes the system specifications not concrete. Therefore, the system constraints may be parameterized, which means that the constraints are defined with parameters.A value of a parameter is chosen from a predefined range at initial state and is fixed for entire execution. It is the case of the real-time protocols and the systems that behave in different ways, according to their configuration.

Using parametric reasoning, it is possible to answer two questions: either verify that a system satisfies a given property for all possible values of parameters, or to find the set of parameters for which the property is satisfied. Many works have been done to develop the theory of parametric reasoning about real time systems [96, 93, 38, 82].

Concrete timing constraints can be expressed and algorithmically verified using real-time temporal logics or timed automata, but to catch the parametric timing constraints the Parametric Timed Automata (PTA) are introduced.

2.2.3.1 Parametric timed automata

Parametric timed automata generalize the timed automata, they characterize a set of parameter values, namely, those for which the automaton has an accepting run.

Definition 10. *A parametric timed automaton is a tuple* $\mathcal{A} = (\Sigma, S, S_0, X, P, \Gamma, F, E)$*, where*

- Σ *is a finite alphabet of events,*

- S *is a finite set of control states,*

- $S_0 \subseteq S$ *is a set of initial control states,*

- $F \subseteq S$ *is a set of accepting (final) control states,*

- X *is a finite set of clocks,*

- P *is a finite set of parameters,*

- $\Gamma \subseteq P$ *is the parameter space, and*

- $E \subseteq S \times G(X,P) \times R \times 2^X \times \Sigma \times S$ *is the set of edges.*

The edges present the transition of the form (s, a, g, r, s'), where s and s' are source and destination location respectively, $a \in \Sigma$ is an action associated with the transition, R is a reset set specifies which clocks are reset, and $g \in G(X,P)$ denotes a transition guard expressed over the clocks and parameters. A pair (ν, p) of a clock valuation and parameter valuation satisfies a guard g, notation $(\nu, p) \models g$, if $q(\nu, p)$ evaluates to true. The *semantics* of a guard g, notation is the set of pairs (ν, p) such that $(\nu, p) \models g$.

Example 3. *The figure illustrates a parametric timed automaton with clocks x, y and parameters α, ψ. The initial state is $S0$ which has invariant $x \leq \alpha$ and the transition from the initial location to $S1$ has guard $y \geq \psi$ and reset set $x := 0$,, and performs the action send. Initially the transition from $S0$ to $S1$ is only enabled if $\alpha \leq \psi$, since the clocks evolve in the same rate. Otherwise the system will be deadlocked.*

The parameterized atomic constraint is an expression in the form:

$$x_i - x_j \bowtie t$$

such that $x_i, x_j \in \mathcal{C}, \bowtie \in (<, \leq, =, \geq, >)$ and $t \in \Gamma(P)$. A set of finite conjunction of parameterized constraints will be noted as $\Omega(\mathcal{C}, \mathcal{P})$. Elements of $\Omega(\mathcal{C}, \mathcal{P})$ are called *parameterized polyhedra*. In this framework, a clock is said to be *nonparametric* if it is never

compared with a parameter, and is *parametric (or parametrically constrained)* otherwise (i.e. if it is compared to a parameter in some transition).

Let γ be an assignment function, $\gamma : P \to \mathbb{N}$ assigning a natural number to each parameter, then $\mathcal{A}^\gamma$ denotes the automaton obtained by setting each parameter $p \in P$ to $\gamma(p)$.

A configuration (s, ν) consists of state s and valuation function $\nu : C \to \mathbb{R}$ that assigns a value to each clock. A transition exists from configuration (s, ν) to (s', ν'), denoted $(s, \nu) \to (s, \nu)$ if either there exists $t \in \mathbb{R}$ such that $\nu'(c) = \nu(c) + t$ for every $c \in C$ or there is an edge $e = (s, R, G, s') \in E$ such that G is satisfied for current clock values and if $c \in R$ then $\nu(c') = 0$ and if $c \notin R$ then $\nu'(c) = \nu(c)$.

A run of a system modeled as PTA is a sequence $\pi = c_1, c_2, ..., c_n$ of configurations such that $c_i \to c_{i+1}$ for each $i < n$. A run is called accepting if c_1 is the initial configuration and is $c_n$in a final state.

2.2.3.2 Emptiness decidability

The problem of testing the emptiness of a language accepted by an automaton is more synthetically called the "emptiness problem". We then say that a model class is decidable if the emptiness problem is decidable for each of these models. We will now present the main results on the decidability of parametric timed automata.

The problem of reachability in parametric timed automata was introduced by Alur, Henzinger, and Vardi [7], it can be expressed as follows: given a timed automaton in which some of the constants appearing within guards on transitions are parameters, is there some assignment of integers to the parameters such that an accepting location of the resulting concrete timed automaton becomes reachable?

A crucial resource of a parametric timed automaton is the number of clocks it employs. Alur, Henzinger, and Vardi [7] showed that, for timed automata with a single parametrically constrained clock (and possibly many concretely constrained clocks), reachability is decidable. In addition, the decidability of reachability for parametric timed automata with two parametric clocks (and arbitrarily many nonparametric clocks) was left open. However, they showed that reachability becomes undecidable for timed automata with three or more parametric clocks. Alur et al. showed that this problem subsumes the question of reachability

in Ibarra's "simple programs" [61] which is open for over 20 years, as well as the decision problem for a fragment of Presburger arithmetic with divisibility.

Concerning the parametric one-counter machines, in [36], the authors show that, in the case of two parametric clocks (and arbitrarily many nonparametric clocks), reachability is decidable for parametric timed automata with a single parameter. In fact, they established a PSPACE^{NEXP} lower bound on the complexity of this problem. Moreover, they have shown in the case of a single parametric clock (with arbitrarily many nonparametric clocks and arbitrarily many parameters) that the reachability problem is NEXP-complete.

Verification of timed automata with parameters is generally undecidable. However, it is decidable for some restricted classes of parametric systems. For instance by bounding the allowed range of the parameters [65] or by requiring that parameters only ever appear either as upper or lower bounds, but never as both [59]: in the latter case, if there is a solution at all then there is one in which parameters are set either to zero or infinity. This latter class is called *lower bound/upper bound (L/U) automata*, it appears to be sufficiently expressive from a practical perspective, while it also has nice theoretical properties. But what is more important is that the emptiness problem is shown to be decidable for L/U automata. The primary concern in such restricted settings is usually the development of practical verification tools, and indeed the resulting algorithms tend to have comparatively good complexity. Moreover, many practical systems outside these classes may be successfully verified using semi-algorithms.

Miller [16] observed that over dense time and with parameters allowed to range over rational numbers, reachability for parametric timed automata becomes undecidable already with a single parametric clock. In the same setting, Doyen [3] showed undecidability of reachability for two parametric clocks even when using exclusively open (i.e., strict) time constraints.

2.2.3.3 Verification and test tools

HyTech: The HyTech tool (Hybrid Tehnology) was developed at Berkeley University by Tom Henziger, Pei-Hsin Hoet and Howard Wong-Toi. This tool aims to verify networks of very general hybrid automata. It even allows parametric systems to be analyzed. It mainly

checks accessibility and safety properties.

UPPAAL: The Uppaal tool was developed jointly by the University of Aalborg in Denmark and the University of Uppsala in Sweden. The main people involved in its development are Wang Yi, Kim G. Larsen, Paul Pettersson, Johan Bengtsson, Gerd Behrmann and Kâre I. Kristoersen. It allows to verify networks of timed automata with bounded integer variables, urgent actions, etc. It mainly checks accessibility and liveliness properties or blocking states. The logic used is a fragment of TCTL. The algorithm implemented is essentially a forward analysis algorithm. Uppaal has a graphical interface that allows the system input (automata) and its execution in an interactive way. In addition, it is possible to generate test cases using Uppaal.

Kronos: The Kronos tool was developed at the Vérimag laboratory in Grenoble mainly by Sergio Yovine, Alfredo Olivero, Conrado Daws and Stravros Tripakis. This tool aims at checking networks of timed automata. It mainly checks properties expressed in the TCTL logic. Several algorithms are implemented: forward analysis and backward analysis.

CMC: The CMC (Compositional Model-Cheking) tool was developed at the Specification and Verification Laboratory at ENS de Cachan by François Laroussinie. This tool aims at checking networks of timed automata. It uses the TCTL logic as a specification language.

IF: The IF (Interchange Format) tool was developed at the Vérimag laboratory in Grenoble, mainly by Jean-Claude Fernandez, Laurent Mounier, Marius Bozga, Luian Ghirvu, Susanne Grafet and Jean-Pierre Krimm. IF is first of all a modeling language for real-time synchronous systems. It has become the link for a set of validation tools, now called the IF validation environment. The IF validation environment is divided into three levels of program representation: the specification level (SDL, OBJECTGEODE,...), the intermediate IF level (SDL2IF, simulation,...) and finally the level of the semantic model LTS (CADP, LTS, verification,...). Finally, it should be noted that the TTG test tools [58] (Timed Test Generation) and TGV [49] (Test Generation using Verification Techniques) are connected to the IF validation environment.

TGV: The TGV tool (Test Generation using Verification techniques) was developed jointly by the Vérimag laboratory in Grenoble and the Irisa laboratory in Rennes. The generation of tests in TGV is oriented "objective of test", i.e. one seeks to generate test cases

for a given behavior of the specification expressed in SDL. TGV uses the commercial tool GEODE (VERILOG) to generate a graph (finite state) of behavior and the tool Aldebaran toolbox CADP [48] to minimize the graph thus obtained. The generation of test trees by TGV is based on a synchronous product as in the process algebra CCS. During the course of the synchronous product, several calculations are made. The algorithm computes the consistency between the specification and the test objective (during a descent phase). A skeleton is synthesized during the ascent phase of the graph. This graph contains sequences of the synchronous product leading to an acceptance state of the automaton. The generated skeletal transitions then decorated with the verdict. Finally, TGV tests the Tretmans *ioco* compliance relationship [62].

TorX: The TorX tool was developed at the University of Twente. TorX is based on the same theory of the test as TGV: similar models and same relation of conformity. The generation algorithm is different since instead of using test objectives for the selection of test cases, TorX randomly draws the inputs of the specification or asks the user to choose one. On the one hand, this simultaneous generation results in TorX only traversing a sequence of observable actions of the specification without having to memorize the states. On the other hand, for each system tested, it is necessary to develop a specific adaptation layer or to pass from the level of abstract tests to executable tests. A version of TorX has been developed in the SPIN tool for Promela specifications.

And they are other tools that rely on different data structures to express the dynamic behavior of the system. For instance, TREX use constrained Parametric DBM for symbolic state representation.

2.3 Abstract interpretation

Code bugs might provoke a dramatic damages, huge financial losses and even human victims. Hence the importance of building correct software and the challenging issue of elaborating formal methods to ensure this correctness. In order to carry out the embedded programs verification and appreciate its importance, it is necessary to have a good understanding of how these programs are designed. In that aim, we shed the light on the so-called "V"

development cycle of critical programs as well as in the techniques put in place during this development to ensure their proper functioning.

Embedded Software Development Cycle:

The most classical (but also most revealing) model of software development is the V cycle. It consists of two phases: the design phase and the integration and verification phase. This latter, which interests us, represents the reconstruction of the overall system from the previously developed components as well as its validation against the specifications stated during the specification phase. The integration phase is also divided into three tasks. The first is when components are developed and consists of verifying each component individually. Formal methods of static analysis are now being used more and more for this purpose. Then, the components are assembled to each other during the construction task of the system, and integration tests are conducted. These include testing communications between system components. Finally, the system validation task manages the functional tests to validate the system against the high-level requirements provided either by the client or by performance constraints.

At this stage, the focus is on testing the system as a whole, and formal methods are rarely used.

It can be noticed that, at different stages of software development, various techniques are used to ensure its proper functioning: the test for the specification and design phases, static analysis (in addition to testing) for the development phase.

Contrary to the test, the static analysis wants to be exhaustive: the goal is to find statically (that is, without executing it) properties that are true for all the executions of the program. These properties can be of different natures. The simplest are surely the properties of safety, where one looks for invariants on the values of the variables of the program, that is to say a range of values X^* as during all executions of the program, the variable x remains in X^*. The form chosen to represent X^* intervals, octagons, polyhedra,. . .) greatly affects the accuracy of this analysis.

Property of another type may also be interesting to prove. Thus, the termination of the program is often an important issue, as is dead code detection (that is, a portion of the program that will never be executed). In the case of reactive programs, there is also a desire

to ensure that the program responds to any signal. Such properties are often categorized as liveness properties, and are considered more difficult to prove: is in this case to prove that the program will have a good behavior.

The concept of abstraction has been used in many different domains from social theory, philosophy to mathematics, and computer science. Generally speaking, abstraction is a generalization process performed by considering only essential information about a concept and neglecting the irrelevant details. In theoretical computer science, Patrick and Radhia Cousot have introduced abstract interpretation, a theory for the abstraction of mathematical structures used in the formal models of computer systems [7]. The purpose of abstract interpretation is to build a sound approximation of the behaviors of undecidable or complex computer systems (i.e. semantics)., used to facilitate the reasoning on and the verification of their behavioral properties (programming language semantics: non-termination, correct termination or with errors, static analysis, formal verification,$\cdots$). Though it was particularly defined for static program analysis, the theory of abstract interpretation has been used in other problems in computer science (model checking, database queries, malware detection) and many other domains [8]. According to the theory of abstract interpretation, the set of all reachable states of a system called the concrete domain are approximated by a reduced and more tractable set called an abstract domain.

2.3.1 Abstract domains

The formal code analysis allows to automatically verify safety of dynamic properties on programs, such as the absence of runtime errors. To achieve this goal, the static analysis should compute the set of reachable program states X in order to be sure that are safe. This is assumed to be done basing on a program semantic function F, such that $F(X) = X$, but unfortunately this is often not computable. To deal with this issue, generally, we define a new abstract semantic function F^*, in order to compute an abstract program invariant X^*, which should include the concrete one X. This technique, so-called abstract interpretation [41, 42], represents an over-approximation of the solution. This over-approximation should be the closest one in order to include less extra-solutions and avoid as possible the false alarms.
A numerical abstraction returns a wider set of reachable states that guarantees the safety

of the result but not its accuracy. So, the stake is to choose the suitable numerical abstract domain and formal methods capable to analyze and cover all program behaviours.

2.3.2 Galois connexion

In abstract interpretation, properties of programs are related between a pair of partially ordered sets: a concrete domain $\mathcal{C} =< D_C, \sqsubseteq >$ and an abstract domain, $\mathcal{A} =< D_A, \preceq >$. Let us explain firstly what partially ordered set means.

Definition 11. *Let S be a set. A relation $\sqsubseteq$ in S is said to be partial order relation if it has the following properties:*

- *if $a \sqsubseteq b$ and $b \sqsubseteq a$ then $b = a$ (anti-symmetry)*
- *if $a \sqsubseteq b$ and $b \sqsubseteq c$ then $b \sqsubseteq c$ (transitivity)*
- *$a \sqsubseteq a$ (reflexivity)*

The pair $(S, \sqsubseteq)$ is said to be partially ordered set (Poset).

The abstract properties are always sound approximations of the concrete properties (abstract proofs/static analyzes are always correct in the concrete) and are sometimes complete (proofs/analyzes of abstract properties can all be done in the abstract only).

When concrete properties all have a $\preceq -$ most precise abstraction the correspondence between the two posets $\mathcal{C}$ and $\mathcal{A}$ is a Galois connection $\mathcal{C} = (D_C, \sqsubseteq) \underset{\alpha}{\overset{\gamma}{\Longleftrightarrow}} (D_A, \preceq)$, formed by a pair of mappings between the domains known as abstraction $\alpha \in \mathcal{C} \rightarrow \mathcal{A}$ and concretization $\gamma \in \mathcal{A} \rightarrow \mathcal{C}$ such that $c \sqsubseteq \gamma(a) \Longleftrightarrow \alpha(c) \preceq a$. ($\Longrightarrow$ expresses soundness and $\Longleftarrow$ best abstraction).

Let $C = (D_C, \sqsubseteq_C)$ and $A = (D_A, \sqsubseteq_A)$ two partially ordered sets. A Galois connection from C to A is a couple $< \alpha, \gamma >$ of functions, where:

$$\forall c \in \mathbb{C}, \forall a \in \mathcal{A} \qquad \alpha(c) \preceq a \Leftrightarrow c \sqsubseteq \gamma(a) \tag{2.15}$$

This equivalence can be written as conjunction of two inclusion invariants:

$$c \sqsubseteq \gamma(\alpha(c)) \bigwedge \alpha(\gamma(a)) \preceq a, \forall c \in \mathcal{C}, \forall a \in \mathcal{A} \tag{2.16}$$

[43]

2.3.3 Abstract interpretation applied to driver behavior analysis

In our laboratory (Laboratoire d'Informatique, Modélisation et Systèmes-LIMS-) we worked on driver behavior analysis. To formally check the compliance of driver to traffic rules we applied the abstraction to its driving data. For driving data abstraction, we propose an adaptation of the interval domain. We choose to use the interval domain because it is more adapted to the models that we will use later. Each collected variable is abstracted separately.

Let $\mathcal{I}_{\bar{\mathbb{R}}} = \{\bot\} \cup \{[a, b[a \in \mathbb{R} \cup \{-\infty\}, b \in \mathbb{R} \cup \{+\infty\}$ and $a < b\}$ the set of right-open intervals with bounds in $\mathbb{R}$ and where $\bot$ is the empty interval. The concrete domain of x is defined as the poset $C_x = <\mathcal{I}_{\bar{\mathbb{R}}}, \subseteq >$ where $\subseteq$ is the usual inclusion operator. Whereas, the abstract domain poset is $A_x =< \mathcal{I}_{\bar{\mathbb{R}}}, \sqsubseteq >$ defines the abstract domain equipped with the ordering relation $\sqsubseteq$. $\sqsubseteq$ is defined by $[a, b[\sqsubseteq [a', b'[\Leftrightarrow a \geq a' and b \leq b'$, and we define the two operators $\sqcap$ and $\sqcup$ by the following:

$X \sqcap Y =\bot$ if $Y =\bot$ or $X =\bot$

$X \sqcap Y =$

In this abstract domain, a set c of reals is approximated with the interval $[a, b[$ where $a = min(c)$ and $b = max(c)$. The abstraction function α_x is thus defined as:

$\alpha_x : \bar{\mathbb{R}} \rightarrow \bar{\mathcal{R}}\ c \mapsto [min(c), max(c)[$ if $c \neq \emptyset$

$\alpha_x(\emptyset) =\bot$

The concretization function γ_x is conversely defined as:

$\gamma_x([a, b[) = \{y \in \mathbb{R} | a \leq y \leq b\}$ and $\gamma_x(\bot) = \emptyset$.

It is easy to see that the functions α_x, γ_x form a Galois connection $< C_x, \subseteq > \underset{\alpha}{\overset{\gamma}{\Longleftrightarrow}} < A_x, \sqsubseteq >$ and we can say then that A_x is a sound approximation of C_x.

Let us take for example the variable "vehicle speed" denoted as *vel*, which values may range from 0 to 240 km/h (the concrete domain). In case of an application that analyzes the compliance of a driver to speed limit panels, *vel* can be abstracted based on the speed limit panels that exist in the traffic code. The French traffic code defines 6 speed limit panels [32]: 30, 50, 70,90,110,130. vel can therefore be abstracted by intervals from the set $A_{vel} = [0, 30[, [30, 50[, [50, 70[, [70, 90[, [90, 110[, [110, 130[, [130, 240[, [240, +\infty)$. Explicitly, all

values of vehicle speed ranging from 0 to 30 will be represented as one abstract value that is the interval $[0, 30[$, thus ignoring the speed changes between 0 and 30 and recording only changes between the significant values for the considered application (abstract values).

The abstract interpretation of driving behavior is defined by the following: Define the concrete and abstract domains: For each driving variable from the vehicle traces, concrete and abstract domains are defined. The concrete domain of a variable V consists of all possible values of V (V=part(IR)). We use an Interval-based abstraction: formally, the abstraction of a variable V, using intervals consists in the approximation of $C_V = x_1, x_2, ..., x_n$ by the least interval [a,b] that enclose them (i.e. a=min(Cv) and b=max(Cv)) $P_V = (C_1, ..., C_n)$ set of concrete values of the variable V $I_V = A_i = [a_i, b_i]|a_i, b_i \in \mathbb{R} and a_i \leq b_i$ the interval domain corresponding to V.

Define the abstraction and concretization functions: Abstraction function: $\alpha : \bar{\mathbb{R}^N} \leftarrow I$

$P_V = (C_1, ..., C_n)$ such that $[min(P_V), max(P_V)] \subseteq [a_i, b_i]$

Concretization function:

$\gamma(A_i = [a_i, b_i]) = \{x \in \mathbb{R} | a_i \leq x \leq b_i\}$

Chapter 3

Constraint Satisfaction Problem

3.1 Introduction

Constraint Satisfaction Problem (CSP) is a fundamental concept in constraints programing, used fundamentally to model and solve research problems, such as optimization, calculus, programming, etc. CSP has received a remarkable interest over the last years, leading to the development of a rich theory that relies on techniques from various areas, especially operation research and artificial intelligence. Most real-world problems can be successfully solved using CSP, among which we can cite resource allocation, scheduling, building design, graph coloring problem, temporal reasoning, financial profits maximization, paths optimization, data clustering, tomography, and more recently natural language processing [16, 33, 97].

Within the CSP framework, a problem is considered as a finite set of variables to which values, satisfying certain problem-specific constraints, are assigned. Actually, solving a CSP aims to achieve one or more of the following goals:

1. Finding all solutions, i.e. all values combinations that satisfy all the constraints.
2. Finding one solution.
3. Detecting an inconsistency.
4. Finding an optimal solution w.r.t. some metrics or objective functions.
5. Finding all optimal solutions.

6. Reducing all interval domains to smaller sizes.

7. Reaching a solved form wherefrom all solutions can be generated easily.

Determining whether a finite CSP (where variables have finite domains) has solution is an NP-complete problem in general [74], which is also the case for finding one solution. An Earlier attempt to solve CSPs follows the guess and check strategy; This latter consists of guessing the assignments of all variables and checking whether they satisfy all constraints. This enables the CSP to be solved in a polynomial time. Actually, a general CSP can be solved in a reasonable time, either by studying the tractability of specific subclasses of the CSP or by using the heuristics and combinatorial search methods. Furthermore, the important result of Schaefer (Dichotomy Theorem) [94] states that every Boolean CSP is contained in one out of six cases and gives necessary and sufficient conditions to classify the problem in polynomial-time or NP-complete. This theorem was recently generalized to a larger class of CSP (i.e. propositional logic of graphs) [28].

Recently, many research have focused on development of effective techniques for CSPs solving, especially for the finite domain case. Examples include Constraint Propagation (CP) [11], Forward Checking (FC) [24], Maintaining Arc Consistency (MAC) [68, 69], and MAC-Backtracking techniques [105]. Another important topic of great application in artificial intelligence, which is considered as a special case of CSP is the Boolean Satisfiability Problem (SAT) [44, 1]. The SAT problem is the first known NP-complete problem, it consists of checking the satisfiability of a given propositional logic formula. Despite its complexity, many of SAT instances that occur in practical issues can be solved in polynomial time. Checking the satisfiability of a formula in Conjunctive Normal Form (CNF) is a SAT subclass where each clause is limited to at most three literals (3-SAT [85]). It is one of Karp's 21 NP-complete problems. On the other hand, 2-SAT and Disjunctive Normal Form (DNF) can be checked in linear time.

Tractability in CSPs can be achieved by considering specific subclasses. These classes are obtained by limiting the allowed domains or relations which can appear in constraints. For example, if the domain is binary and all variables are binary, satisfiability is polynomial-time solvable (equivalent to 2-SAT). In our side, we deal with a subclass of CSPs, where

constraints are expressed as inequalities written in one of the following forms: $\{ x \sim \alpha, x-y \sim \beta, (x-y)-(z-t) \sim \lambda\}$, such that x, y, z and t are real variables, α, β and λ are real constants and $\sim \in \{\leq, \geq\}$. We denote this CSP subclass by "4-CSP".

3.2 Constraint language

Definition 12 (Constraint Satisfaction Problem)**.** *A Constraint Satisfaction Problem is a triplet $N = (X, D, C)$, where:*

- $X = \{x_1, x_2, ..., x_n\}$ *is a set of variables.*
- $D = D_{x_1} \times D_{x_2} \times ... \times D_{x_n}$ *is the domain for X, where $D_{x_i} \in \mathbb{T}$ is the set of possible values for the variable x_i.*
- $C = \{c_1, c_2, ..., c_r\}$ *is a set of constraints.*

A constraint $c_i \in C$ is a pair $< u_i, R_i >$, where $u_i \subseteq X$ is subset of k variables and R_i is a k-ary relation on these variables. A valuation ν satisfies $< u_i, R_i >$ if the values assigned to the variables of u_i satisfy the relation R_i. A valuation is consistent if it verifies all the constraints in C (i.e. $\bigwedge c_i$), and is complete if it includes all variables. Each valuation that is consistent and complete is a CSP solution. By abuse of notation, $\bigwedge c_i$ denotes the CSP constraint set, and we write $C = \bigwedge c_i$.

Let C and T be two finite relational structures over the same vocabulary. A homomorphism from S to T is a mapping from the elements of S to elements of T such that all elements related by some relation C_i in S map to elements related by C_i in T. if T is a restriction of S to a subset of the elements, and the homomorphism from S to T is just the identity mapping when restricted to T, then the homomorphism is called retraction, and T is called a retract of S.

A constraint satisfaction problem will be here a problem of the following form. The first finite relational structure T over some vocabulary is called template. An instance is a finite relational structure S over the same vocabulary. The instance is satisfied if there is a homomorphism from S to T. Such a homomorphism is called a solution.

In a constraint satisfaction problem (CSP) the question is to decide whether or not it is possible to satisfy simultaneously a given set of constraints. One of the formal ways to specify a constraint is to express them as set of relations. Each combination of variables belongs to a given relation. If all the combinations belong to the same relation, then the problem is referred to as a uniform CSP. Otherwise, if they come from some set γ of relations the problem is said to be nonuniform. The nonempty set of relations γ is then called a constraint language if there exists an assignment $I : V - - >$, such that I satisfies every $C \in \phi$ For example, consider 3SAT: a well-known restriction of the general satisfiability problem. 3SAT can be seen to be the CSP over the language

Many different variations of the CSP have been studied across various fields. These matters include counting CSPs, optimization CSPs, quantified CSP, valued CSP,etc. More details about CSP studies and the survey of the state-of-the art can be found in this recent book [67].

A systematic study of the complexity of nonuniform CSPs was started by Schaefer in 1978 [95] who classified the Boolean constraint satisfaction problem and showed that, for every constraint language γ over a 2-element set the problem CSP(γ) is either solvable in polynomial time or is NP-complete, depending on the allowed relations in the propositional formula. He showed that there are only three polynomially solvable constraint satisfaction problem on the set {0, 1} namely 2SAT, Horn clauses and linear equations modulo 2; all CSP on {0, 1} that do not fit into one of these three categories are NP-complete.

The next step in the study of nonuniform CSPs was made in the seminal paper by Feder and Vardi [47], who apart from considering numerous aspects of the problem, posed the Dichotomy Conjecture that states that for every finite constraint language Γ over a finite set the problem CSP(γ) is either solvable in polynomial time or is NP-complete. They set up three classes: bounded width, bounded strict width and subgroup, as generalization of the ones of Schaefer. In fact, a CSP is said to have bounded width if its complement (i.e., the non-existence of a solution) can be expressed in Datalog. More precisely, it is said to have width (l,k) if the corresponding Datalog program has rules with at most l variables in the head and at most k variables per rule, and it is said to have width l if it has width (l,k) for some k. Note that the Datalog is the language of logic programs without function symbols.

It is worth mentioning that this Dichotomy Conjecture is proved by A. Bulatov [35]. A CSP is group-theoretic when the elements of the template T are elements in a finite group H.

3.3 Algorithms

Constraint satisfaction problems on finite domains are typically solved using a form of search. The most used techniques are variants of backtracking, constraint propagation, and local search.

Backtracking is a recursive algorithm. It maintains a partial assignment of the variables. Initially, all variables are unassigned. At each step, a variable is chosen, and all possible values are assigned to it in turn. For each value, the consistency of the partial assignment with the constraints is checked; in case of consistency, a recursive call is performed. When all values have been tried, the algorithm backtracks. In this basic backtracking algorithm, consistency is defined as the satisfaction of all constraints whose variables are all assigned. Several variants of backtracking exist. Backmarking improves the efficiency of checking consistency. Backjumping allows saving part of the search by backtracking "more than one variable" in some cases. Constraint learning infers and saves new constraints that can be later used to avoid part of the search. Look-ahead is also often used in backtracking to attempt to foresee the effects of choosing a variable or a value, thus sometimes determining in advance when a subproblem is satisfiable or unsatisfiable.

Constraint propagation techniques are methods used to modify a constraint satisfaction problem. More precisely, they are methods that enforce a form of local consistency, which are conditions related to the consistency of a group of variables and/or constraints. Constraint propagation has various uses. First, it turns a problem into one that is equivalent but is usually simpler to solve. Second, it may prove satisfiability or unsatisfiability of problems. This is not guaranteed to happen in general; however, it always happens for some forms of constraint propagation and/or for certain kinds of problems. The most known and used form of local consistency are arc consistency, hyper-arc consistency, and path consistency. The most popular constraint propagation method is the AC-3 algorithm, which enforces arc consistency.

Local search methods are incomplete satisfiability algorithms. They may find a solution of a problem, but they may fail even if the problem is satisfiable. They work by iteratively improving a complete assignment over the variables. At each step, a small number of variables are changed value, with the overall aim of increasing the number of constraints satisfied by this assignment. The min-conflicts algorithm is a local search algorithm specific for CSPs and based in that principle. In practice, local search appears to work well when these changes are also affected by random choices. Integration of search with local search have been developed, leading to hybrid algorithms.

CSPs are also studied in computational complexity theory and finite model theory. An important question is whether for each set of relations, the set of all CSPs that can be represented using only relations chosen from that set is either in P or NP-complete. If such a dichotomy theorem is true, then CSPs provide one of the largest known subsets of NP which avoids NP-intermediate problems, whose existence was demonstrated by Ladner's theorem under the assumption that $P \neq NP$. Schaefer's dichotomy theorem handles the case when all the available relations are Boolean operators, that is, for domain size 2. Schaefer's dichotomy theorem was recently generalized to a larger class of relations.

Most classes of CSPs that are known to be tractable are those where the hypergraph of constraints has bounded treewidth (and there are no restrictions on the set of constraint relations), or where the constraints have arbitrary form but there exist essentially non-unary polymorphisms[clarification needed] of the set of constraint relations. Every CSP can also be considered as a conjunctive query containment problem.

Chapter 4

4-Constraint Satisfaction Problem tractability

The whole of this chapter and the next one present the fruit of our made efforts. We provided a strong proofs for all theorems. As confirmed, determining whether a finite CSP (where variables have finite domains) has solution is an NP-complete problem in general [74], which is also the case for finding one solution. There are several reasons for which studying CSPs is important, firstly because CSP are omnipresent in a number of real-world problems and secondly for their reduced complexity proven to be polynomial. Many types of hard and useful real-world problems can be modeled as 4-CSPs. The Four Phase Handshake Protocol [27] given in [66], and depicted in Figure 4 is one example, amongst others. This protocol uses two clocks x_1, x_2, two parameters $minIO$, $maxIO$, and the following constraints: $(x_1 < maxIO)$, $(x_1 > inIO)$, $(x_2 < maxIO)$, $(x_2 > inIO)$, and $((x_1 - x_2) \leq (maxIO - minIO))$. Dealing with the protocol comes down to deal with its equivalent 4-CSP.

An other relevant example that shows the utility of 4-CSPs is the expression of temporal constraints in real-time systems. Almost all the systems modeled by Parametric Timed Automata (PTA)[7], one of the most popular formalisms for modeling real-time systems, can be represented by a 4-CSP. PTA facilitates the manipulation of real-time systems, especially for their control and verification. Unfortunately, most PTA verification problems are unde-

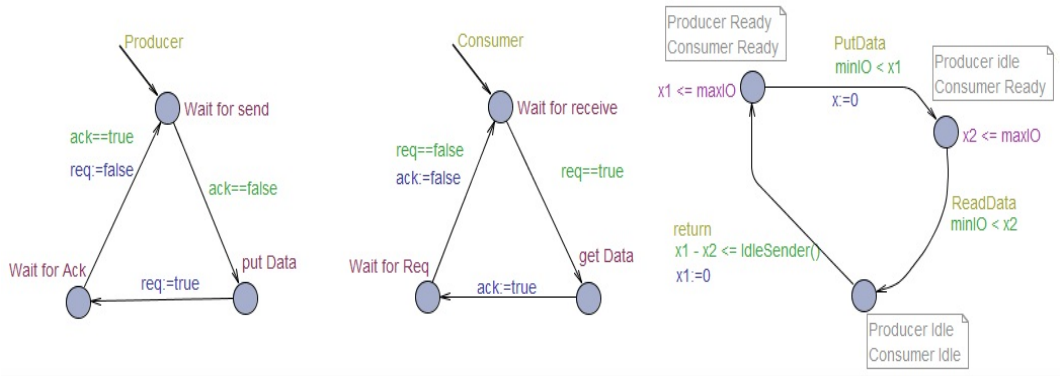

Figure 4.1: 4-phase handshake protocol

cidable [19, 9]. In PTA, a clock (timer) or a difference of two-clocks is compared to a linear combination of parameters. Hune et al. [60] identify the subclass "lower bound/upper bound (L/U) automata", where each parameter occurs in the timing constraints either as a lower bound or as an upper bound. In fact, L/U automata can be used to model the Fisher's mutual exclusion algorithm, the root contention protocol, and other known examples from the literature (see [60]). The algorithms based on the 4-CSP framework can serve for an accelerated L/U automata verification.

In addition, the results presented in this chapter show that the 4-CSP can be used to derive a numerical abstract domain [41, 42, 53]. Numerical abstract domains are widely used in static program analysis. Numerical abstraction [41, 45, 78, 39, 54], applied in static code analysis, provides a wider set of reachable states that guarantees the safety of the result. The challenge consists of choosing the suitable numerical abstract domain and formal methods capable to analyze all program behaviors. The abstract domain defined based of the 4-CSP, in this case, extends the conjunction of octagonal invariants [78], called Unit Two Variable Per Inequality (UTVPI) constraints, with inequalities of the form: $\{(x - y) - (z - t) \leq \lambda\}$. The precision of the derived domain lies between the domains of octagons [78] and polyhedra [54].

4.1 Problem statement

Using the general CSP framework to express the constraints in the aforementioned examples is very complex and almost NP-Complete. The 4-CSP framework is however precise enough to cover all their constraints and has the advantage to be linear in time and space. The 4-CSP constraint set can be seen as a subclass of the octahedron constraint set [39] which has the form: $\Sigma(x_i) - \Sigma(x_j) \geq k, k \in \mathbb{Q}$. However, the complexity of octahedra operations over n variables is 3^n in memory and 3^n in execution time [39], which is very costly compared to the complexity of our implementation proved to be cubic in the number of variables. One question that immediately comes to mind when solving the 4-CSP is whether to simply use Linear Programming (LP) classical solving techniques. Actually, the LP seems to be not suitable for the 4-CSP presented in this part. This is because LP aims to find an optimal solution of the CSP with regard to some objective function, while the computation methods based on 4-CSP framework have many other goals: to guarantee the existence of a solution, to reduce all interval domains to smaller sizes and to achieve a solved form wherefrom all solutions can be generated easily.

To sum up, the main contributions of this part consist of the following:

1. Setting a theoretical basis for the 4-CSP and giving a data structure for its domains.

2. Providing, based on hypergraph theory coupled with positive linear dependence theory, to the best of our knowledge, the first graph-based method for the 4-CSP tractability.

3. Developping a modified arc consistency algorithm that combines the MAC algorithm with the hypergraph closure in order to easily solve the 4-CSPs: either to check the emptiness of solution set or to list the solutions. All these operations have a polynomial time and space complexity.

In the remaining of this chapter, we set up some basic definitions and we provide the mathematical proved background of the part. In fact, we establish a hypergraph-based characterization of the feasibility problem.

4.2 Some results on Positive Linear Dependence

Throughout the following sections, we use the following notations:

- $\mathbb{N}$ (resp. $\mathbb{Z}$) denotes the set of natural numbers (resp. integers) and $\mathbb{R}$ (resp. $\mathbb{Q}$) the set of real (resp. rational) numbers.
- For a domain $\mathbb{T}$ ($\mathbb{R}$ or $\mathbb{Q}$), and $n \in \mathbb{N}$:
  - $\mathbb{T}^{\geq 0}$ denotes the set $\{x \,|\, x \geq 0, x \in \mathbb{T}\}$. $+\infty$ (resp. $-\infty$) denotes positive (resp. negative) infinity such that: for all $t \in \mathbb{T}$, $-\infty < t < +\infty$, $t+(+\infty) = (+\infty)+t = +\infty$ and $t + (-\infty) = (-\infty) + t = -\infty$. $\overline{\mathbb{T}}$ denotes $\mathbb{T} \cup \{+\infty, -\infty\}$.
  - $\mathcal{P}(\mathbb{T})$ denotes the power set of $\mathbb{T}$. For a set $S \subseteq \mathcal{P}(\mathbb{T})$, $min(S)$ (resp. $max(S)$) is the minimal (resp. maximal) element of S. When S has no lower bound (resp. upper bound), then $min(S) = -\infty$ (resp. $max(S) = +\infty$).
  - For a set $X = \{x_1, x_2, ..., x_n\}$ of valued variables over $\mathbb{T}$, a valuation ν over X is a function that associates to each variable of X, a value in $\mathbb{T}$:

$$\begin{array}{rccc} \nu: & X & \longrightarrow & \mathbb{T} \\ & x_i & \longmapsto & \nu_i \end{array}$$

    ν can be seen as a vector of $\mathbb{T}^n$. $\mathcal{V}(X)$ denotes the set of valuations over X.

    D_{x_i} is the set of possible values for the variable x_i and it is called domain of x_i. x_0 is a special variable that is always equal to zero i.e. $D_{x_0} = \{0\}$ and $X^0 = X \cup \{x_0\}$.
  - $(\mathbb{T}^n, +, \times)$ denotes the n-dimensional vector space over $\mathbb{T}$. The vector $\boldsymbol{e_0}$ denotes the zero vector of $\mathbb{T}^n$. The set $\{\boldsymbol{e_1}, \boldsymbol{e_2}, \cdots, \boldsymbol{e_n}\}$ denotes the canonical basis (standard basis) of $\mathbb{T}^n$, that is :

$$\text{For all } \boldsymbol{e_i} = (a_j)_{j\in[1,n]} \Rightarrow \begin{cases} a_j = 1 & \text{If } j = i \\ a_j = 0 & \text{Otherwise} \end{cases}$$

The theory of positive linear dependence was initiated by J. Farkas[46] and T. Motzkin[79], and developed by Chandler Davis [37]. In this part, we consider an adaptation of this theory.

Therefore, the definitions given in the rest of this section are a slightly different from those of Chandler. After giving the adapted definitions, the fundamental theorem for the simple, positively dependent sets is introduced.

Let $f = (V_i)_{i\in[1,r]}$ be a family of distinct, non-empty vectors of $\mathbb{T}^n$. A *strictly positive combination* of f is a linear combination $\sum_{i=1}^{r} \lambda_i V_i$, with $\lambda_i \in \mathbb{N}^{>0}$.

Definition 13. *f is said to be **positively independent** if none of the strictly positive combinations of f is equal to $\boldsymbol{e_0}$. Otherwise, f is **positively dependent** (ie. there exist some scalars $\lambda_i \in \mathbb{N}^{>0}$ such that $\sum_{i=1}^{r} \lambda_i V_i = \boldsymbol{e_0}$). A positively dependent family f is said to be **simple**, if every subfamily $f' \subset f$ is positively independent.* □

Theorem 6. *If f is simple, then the scalars $(\lambda_i)_{i\in[1,r]} \in \mathbb{N}^{>0}$ that satisfy the equation $\sum_{i=1}^{r} \lambda_i V_i = \boldsymbol{e_0}$ are unique (up to multiplication by a positive constant). The unique (minimal) solution is denoted by $U(f)$.* □

In other words, if we have $(\lambda_i)_{i\in[1,r]} \in \mathbb{N}^{>0}$ and $(\alpha_i)_{i\in[1,r]} \in \mathbb{N}^{>0}$ such that $\sum_{i=1}^{r} \lambda_i V_i = \sum_{i=1}^{r} \alpha_i V_i = \boldsymbol{e_0}$, then $\frac{\alpha_i}{\lambda_i} = \frac{\alpha_j}{\lambda_j}$ for all $i, j \in [1, r]$. This can be proved based on the proof of "theorem 4.3" given by Chandler in [37].

Proof. The proof is based on the following claim:

- For all $1 \leq p < r$, the only scalars $(\alpha_i)_{i\in[1,p]} \in \mathbb{Z}$ that satisfy the equation $\sum \alpha_i V_i = \boldsymbol{e_0}$, are $\alpha_i = 0$ for $i \in [1, p]$

This claim states that any sub-family of f is not positively independent in $\mathbb{Z}$. Since f is positively dependent, then, there exist $(\lambda_i)_{i\in[1,r]} \in \mathbb{N}^{>0}$, such that

$$\lambda_1 \times V_1 + \lambda_2 \times V_2 + \cdots + \lambda_p \times V_p + \cdots + \lambda_r \times V_r = \boldsymbol{e_0} \tag{4.1}$$

Now, assuming that we can find $(\alpha_i)_{i\in[1,p]} \in \mathbb{Z}$ such that $\alpha_i \neq 0$, for all $i \in [1, p]$ and:

$$\alpha_1 \times V_1 + \alpha_2 \times V_2 + \cdots + \alpha_p \times V_p = \boldsymbol{e_0} \tag{4.2}$$

And, let $m_j = min(\{\frac{\lambda_i}{|\alpha_i|} \,|\, i \in [1, p], \alpha_i < 0\})$ be the minimal value reached by $\frac{\lambda_j}{|\alpha_j|}$.

We can therefore deduce that:

$$\lambda_i + m_j \times \alpha_i = \lambda_i + \frac{\lambda_j}{|\alpha_j|} \times \alpha_i = \lambda_j(\frac{\lambda_i}{\lambda_j} + \frac{\alpha_i}{|\alpha_j|})$$

It is clear that this sum is grater that zero if $\alpha_i > 0$.

If $\alpha_i < 0$, since

$$m_j = \frac{\lambda_j}{|\alpha_j|} \leq \frac{\lambda_i}{|\alpha_i|} \text{ implies that } \frac{|\alpha_i|}{|\alpha_j|} \leq \frac{\lambda_i}{\lambda_j}. \text{ Thus } 0 = (\frac{|\alpha_i|}{|\alpha_j|} + \frac{\alpha_i}{|\alpha_j|}) \leq (\frac{\lambda_i}{\lambda_j} + \frac{\alpha_i}{|\alpha_j|})$$

From this, we can conclude that $\lambda_i + m_j \times \alpha_i \geq 0$ for all $i \in [1, p]$ and $\lambda_j + m_j \times \alpha_j = 0$. Finally, by multiplying equation (4.2) with the positive scalar m_j and adding it to equation (4.1), we end up with the following new equation:

$$(\lambda_1+m_j\times\alpha_1)\times V_1+\cdots+(\lambda_j+m_j\times\alpha_j)\times V_j+\cdots+(\lambda_p+m_j\times\alpha_p)\times V_p+\lambda_{p+1}\times V_{p+1}+\cdots+\lambda_r\times V_r = \boldsymbol{e_0} \tag{4.3}$$

$\lambda_j + m_j \times \alpha_j = 0$ means that there is a sub-family of f which is positively dependent. This appears to contradict the fact that f is simple.

Now, if we have $(\lambda_i)_{i\in[1,r]} \in \mathbb{N}^{>0}$ and $(\alpha_i)_{i\in[1,r]} \in \mathbb{N}^{>0}$ such that $\sum_{i=1}^{r} \lambda_i V_i = \sum_{i=1}^{r} \alpha_i V_i = \boldsymbol{e_0}$, then it is easy to see that $\alpha_r \times \sum_{i=1}^{r} \lambda_i V_i - \lambda_r \times \sum_{i=1}^{r} \alpha_i V_i = \sum_{i=1}^{r}((\alpha_r \times \lambda_i) - (\lambda_r \times \alpha_i))V_i = \boldsymbol{e_0}$ and has at most $r-1$ vectors. From the previous result we deduce that: $\lambda_r \times \alpha_i = \alpha_r \times \lambda_i$, for all $i \leq r$. □ □

In this manner, Theorem 6 states a fundamental result that allows the characterization of constraints to be considered while checking the emptiness of a general CSP. Furthermore, it identifies the set of constraints that may have an impact on the computation of the tight bound of a given linear constraint. The case of 4-CSP is further explained in the section 2.3.

4.3 4-Constraint Satisfaction Problem tractability

Most of research works dealing with CSPs consider binary constraints (i.e. $k = 2$). The constraints considered in this work are defined in the next paragraphs to be atomic 4-Constraints.

Motivated by many real-life problems like timed system verification, we introduce the 4-CSP with the atomic 4-constraints defined below. Let $X = \{x_1, x_2, ..., x_n\}$ be a set of real-valued variables over $\mathbb{T}$.

Definition 14. *An **atomic 4-constraint** over X is an inequality of the form:*

$$(\epsilon_i x_i - \epsilon_j x_j) - (\epsilon_p x_p - \epsilon_q x_q) \sim m_{ijpq}$$

where $m_{ijpq} \in \mathbb{T}$, $\sim \in \{\leq, \geq\}$, and for all $k \in \{i, j, p, q\}$, $\epsilon_k \in \{0, 1\}$.

*An atomic 4-constraint is said to be in its **canonical form** iff for all $k \in \{i, j, p, q\}$, $\epsilon_k \neq 0$ and " $\sim$ " is equal to " $\leq$ ".* □

For instance, $(x_1+x_2-x_3-x_4 \leq 4)$, $(x_1+x_2 \leq 5)$, $(x_1+x_2-x_3 \leq 6)$ and $(x_1-x_2-x_3 \leq 8)$ are atomic 4-constraints. It is easy to see that, by introducing a special variable x_0, which is always equal to zero, every atomic 4-constraint might be converted to its canonical form. For example, the 4-constraint $(0 \times x_i - 1 \times x_j) - (1 \times x_p - 1 \times x_q) \sim m_{ijpq}$ can be written as: $(x_0 - x_j) - (x_p - x_q) \sim m_{ijpq}$.

The set of atomic (resp. canonical) 4-constraints over X is denoted by $\Phi(X)$ (resp. **4-**$\Phi(X)$). In the rest of the part, we will not distinguish between $\Phi(X)$ and 4-$\Phi(X^0)$, and we will consider only canonical 4-constraints. For a canonical 4-constraint $c_{ijpq} = (x_i - x_j) - (x_p - x_q) \leq m_{ijpq}$, we define:

- The **normal vector** of the hyperplane induced by c_{ijpq} (variables involved in c_{ijpq}):

$$\begin{array}{rccc} F_v: & 4-\Phi(X^0) & \longrightarrow & I^n \\ & (x_i - x_j) - (x_p - x_q) \leq m_{ijpq} & \longmapsto & \boldsymbol{e_i - e_j - e_p + e_q} \end{array}$$

- The **upper bound** (**the weight function**):

$$\begin{array}{rccc} F_b: & 4-\Phi(X^0) & \longrightarrow & \mathbb{T} \\ & (x_i - x_j) - (x_p - x_q) \leq m_{ijpq} & \longmapsto & m_{ijpq} \end{array}$$

- The **complement** :

$$\overline{c_{ijpq}} = c_{jiqp} = ((x_j - x_i) - (x_q - x_p) \leq m_{jiqp})$$

Note that, $F_v(c) = -F_v(\overline{c})$ for every constraint $c \in 4\text{-}\Phi(X^0)$.

Definition 15. *A **4-CSP** S over X, is expressed as a conjunction of constraint set noted:*

$$C_s = \bigwedge((x_i - x_j) - (x_p - x_q) \leq m_{ijpq})$$

A solution of the 4-CSP is then a solution of m canonical 4-constraints over X^0, where m is the number of non-redundant conjunction terms. □

For a 4-CSP S over X, we denote by C_s the set of all canonical 4-constraints of S, and D_s the domain of solutions for $4 - CSP$. For a valuation $\nu \in \mathcal{V}(X^0)$, $\nu \in D_s$ iff ν satisfies all constraints of C_s. D_s is an empty set iff for all $\nu \in \mathcal{V}(X^0)$, $\nu \notin D_s$. As an example, the 4-CSP defined by the following 4-constraints:

$$C_s = (x_1 + x_0 - x_2 - x_3 \leq 3) \wedge (x_2 + x_0 - x_1 - x_4 \leq -4) \wedge (x_4 + x_3 - x_0 - x_0 \leq 5) \bigwedge$$

$$(x_2 + x_0 - x_0 - x_0 \leq 3) \wedge (x_3 + x_0 - x_0 - x_0 \leq 1) \wedge (x_4 + x_0 - x_0 - x_0 \leq 5) \wedge (x_1 + x_0 - x_0 - x_0 \leq 6)$$

is not empty since the valuation defined by $(x_0, x_1, x_2, x_3, x_4) = (0, 6, 3, 1, 2) \in D_s$.

In order to keep bounds of constraints involved in the 4-CSP S, we extend the mapping F_b to C_s, in the usual way:

$$F_b^s : \begin{array}{ccc} 4 - \Phi(X^0) & \longrightarrow & \overline{\mathbb{T}} \\ c & \longmapsto & \begin{cases} F_b(c) & \text{If } c \in C_s \\ +\infty & \text{Otherwise} \end{cases} \end{array}$$

In this way, F_b^s keeps the upper bounds of constraints involved in S and sets to positive infinity the other constraints not in C_s (the weight function related to S). Finally, S is said

to be a **bounded 4-CSP** if there exits a scalar $w \in \mathbb{T}$ such that:

$$D_s \subseteq \{\nu \,|\, \nu \in \mathcal{V}(X) \text{ such that } -w < \nu_i < w\}$$

4.4 Hypergraph based characterization of the tractability problem

Graph-based algorithms has been widely used for checking the feasibility (or the emptiness) of a system of inequalities with restricted form such as the potential constraints conjunctions [45] ($\bigwedge(x_i - x_j \leq m_{ij})$) and Octagons [78] ($\bigwedge(\pm x_i \pm x_j \leq m_{ij})$). In the case of potential constraints, a data structure called Difference Bound Matrices (DBM) is used to store the system constraints. A DBM can be seen as the adjacency matrix of a directed graph $G = (N, E, w)$ (potential graph), where the set N corresponds to the system variables, $E \subseteq N^2$ and $w \in E \mapsto \mathbb{T}$ is the weight function defined by:

$$\begin{cases} (x_i, x_j) \notin E & \text{if } m_{ij} = +\infty, \\ (x_i, x_j) \in E \text{ and } w(x_i, x_j) = m_{ij} & \text{if } m_{ij} \neq +\infty \ . \end{cases}$$

A well known result of Bellman [18] shows when DBMs are feasible. In fact, Bellman proves that a DBM is empty if and only if there exists, in its associated potential graph, a cycle with a strictly negative total weight. The concept of cycles (either simple cycle or closed walk) used in graph theory is able to handle constraints of the form $\pm x_i \pm x_j \leq m_{ij}$ (plan constraints). However, it will not handle constraints of the form $(x_i - x_j) - (x_p - x_q) \leq m_{ijpq}$ (hyperplane constraints).

Broadly speaking, this work aims to develop scalable algorithms based on graph theory, for the feasibility checking and canonical form computation of CSPs. The question that immediately arises is can a graph theory based approach for general CSP feasibility characterization achieve similar results to that of Bellman? As will be discussed, the answer is fortunately positive for 4-CSP. This is because hypergraph theory coupled with positive

linear dependence theory gives us strong theoretical tools to answer the raised question. In this part, we restrict ourselves to 4-CSP; however, the results can be extended to CSP with constraints similar to those represented by the Octahedra abstract domain[39].

Definition 16. *A directed hypergraph [13] H is a pair (N, E), where N is a non empty set of nodes and E is a set of hyperarcs. A hyperarc e is an ordered pair (T, h), with $T \subseteq N$, $T \neq \emptyset$, and $h \in N \backslash T$. T and h are called the tail and the head of e, and are denoted by $tail(e)$ and $head(e)$, respectively. A weighted directed hypergraph (N, E, w), is a directed hypergraph (N, E) that has a positive number $w(e)$ associated with each hyperarc e, called the weight of hyperarc e.* □

Clearly, a 4-CSP S over X can be easily mapped to a weighted directed hypergraph (N, E, w). In fact, the set of nodes N will correspond to the set of variables X^0. Each constraint $c_{ijpq} \in C(S)$ defines the hyperarc $e = (T, h)$ such that: $T = \{x_j, x_p\}$ and $h = \{x_i, x_q\}$. In other words, the normal vector $F_v(c_{ijpq})$ of c_{ijpq}, can be mapped to a unique hyperarc: positive values of $F_v(c_{ijpq})$ are mapped to the head of e, and negative values to the tail of e. Furthermore, we can associate a weight function to the hypergraph defined by $w(e) = F_b^s(c_{ijpq})$.

Since the first parts of Berger [21], hypergraph theory has been a useful tool in several fields including computer science, mathematics, bio-informatics, engineering, and chemistry [106]. Since a hypergraph is nothing but a family of sets, and for the sake of clarity, in this part, we will use the terminology of the hypergraph theory together with the notations of positive linear dependence theory. Thus, **rather than using a hyperarc to map a 4-constraint, we use the corresponding normal vector** $F_v()$, and we extend the notions of paths, cycles and minimal weights to hypergraphs in a consistent manner.

4.4.1 Hypercycles and hyperpaths

Definition 17. *Let $C = \{c_1, c_2, \cdots, c_r\}$ be a set of distinct constraints of 4-$\Phi(X^0)$. We say that C generates a* ***hypercycle*** *(**h-cycle** for short) if the family $f = (F_v(c_i))_{i \in [1,r]}$ of normal vectors is positively dependent. We say that C generates a* ***simple hypercycle*** *if*

$f = (F_v(c_i))_{i\in[1,r]}$ is simple positively dependent. $\square$

Intuitively, C generates a hypercycle if we can find some strictly positive natural numbers λ_i such that the sum $\sum \lambda_i F_v(c_i)$ equals the empty vector. On the one hand, this definition is quite different from those found in the literature in the sense that, the h-cycle nodes are required to appear as hyperarc tails the same number of times they appear as hyperarc heads in the associated hypergraph. On the other hand, hypercycles can be seen as a generalization of graph-based cycles where (λ_i) are equal to 1. In fact, each edge (x_i, x_j) of a cycle in a graph defines the normal vector $V_{ij} = e_i - e_j$. One can notice that $\sum 1 \times V_{ij} = \sum (e_i - e_j)$ equals the zero vector. Thus, the family (V_{ij}) is positively dependent, which means that the set $C = \{c_1 = (x_i - x_j \leq w(x_i, x_j)), c_2 = (x_j - x_l \leq w(x_j, x_l)), \cdots, c_k = (x_k - x_i \leq w(x_k, x_i))\}$ generates a h-cycle. Regarding the simple h-cycle, it is the hypercycle that can not be decomposed into multiple hypercycles (like elementary cycle in graphs). Note that, the set $\{c, \overline{c}\}$ generates a simple h-cycle for every constraint $c \in 4\text{-}\Phi(X^0)$.

For instance, assuming that $X = \{x_1, x_2, x_3, x_4\}$:

1. The set $C = \{c_1 = (x_1 + x_0 - x_2 - x_3 \leq 3), c_2 = (x_2 + x_0 - x_1 - x_4 \leq -4), c_3 = (x_4 + x_3 - x_0 - x_0 \leq 5)\}$ generates a h-cycle as $F_v(c_1) = (1, -1, -1, 0)$, $F_v(c_2) = (-1, 1, 0, -1)$, $F_v(c_3) = (0, 0, 1, 1)$ and $F_v(c_1) + F_v(c_2) + F_v(c_3) = \mathbf{0}$.

2. The set $C = \{c_1 = (x_1 + x_0 - x_2 - x_3 \leq 3), c_2 = (x_1 + x_2 - x_3 - x_0 \leq -4), c_3 = (x_0 + x_3 - x_1 - x_0 \leq 5)\}$ generates a h-cycle as $F_v(c_1) = (1, -1, -1, 0)$, $F_v(c_2) = (1, 1, -1, 0)$, $F_v(c_3) = (-1, 0, 1, 0)$ and $F_v(c_1) + F_v(c_2) + 2 \times F_v(c_3) = \mathbf{0}$.

In the remaining, the set of all hypercycles over $4\text{-}\Phi(X^0)$ will be denoted:

$$HCycle(X^0) = \{(C, (\lambda_i)_{i\in[1,r]}) \,|\, C = \{c_1, c_2, \cdots, c_r\},\ (\lambda_i)_{i\in[1,r]} \in \mathbb{N}^{>0},\ \text{and}\ \sum_{i=1}^{r} \lambda_i F_v(c_i) = \boldsymbol{e_0}\}$$

In a similar way, the notion of graph paths can be extended to **hyperpaths** as follows:

Definition 18. *Let $P = \{c_1, c_2, \cdots, c_r\} \subseteq 4\text{-}\Phi(X^0)$, and $c \in 4\text{-}\Phi(X^0)$. Then, P generates a* ***hyperpath*** *(**h-path** for short) of c, if $P \cup \{\overline{c}\}$ generates a hypercycle. P generates a* ***simple hyperpath*** *of c, if $P \cup \{\overline{c}\}$ generates a simple hypercycle.* $\square$

From the previous example, it is easy to see that $\{(x_1+x_0-x_2-x_3 \leq 3), (x_2+x_0-x_1-x_4 \leq -4)\}$ generates a hyperpath of $(x_0+x_0-x_3-x_4 \leq 6)$. The set of all hyperpaths of $c \in 4\text{-}\Phi(X^0)$ will be denoted by:

$$HPath(c) = \{(P, (\frac{\lambda_i}{\lambda})_{i\in[1,r]}) \,|\, P = \{c_1, c_2, \cdots, c_r\},\ (\lambda, \lambda_i) \in \mathbb{N}^{>0} \text{ and } F_v(\overline{c}) + \sum_{i=1}^{r} \frac{\lambda_i}{\lambda} F_v(c_i) = \boldsymbol{e_0}\}$$

Remark 1. *As mentioned before, each 4-constraint generates a unique hyperarc and thus the definitions 17 and* **??** *hold for the hypergraph associated to the 4-CSP.*

4.4.2 Some results on positive hypercycles

Let S be a 4-CSP over X and $H^s = (N, E, w)$ the weighted directed hypergraph associated to S. As is the case with weighted graphs, S defines the minimum weight hypergaph $H^s_m = (N, E, w_m)$. Before defining H^s_m, let us extend the weight function $F^s_b()$ (resp. w) of S (resp. of H^s) to hypercycles and hyperpaths, in the usual way:

Definition 19. *Let $c \in 4-\phi(X^0)$ be a 4-constraint. Then:*

- *For a h-path $(P, (\lambda_i)) \in HPath(c)$ of c such that $P = \{p_1, p_2, \cdots\}$, the weight of P in S (and it is the same for H^s) is: $w((P, (\lambda_i))) = F^s_b((P, (\lambda_i))) = \sum \lambda_i F^s_b(p_i)$.*
- *For a h-cycle $(C, (\alpha_i)) \in HCycle(X^0)$ such that $C = \{c_1, c_2, \cdots\}$, the weight of C in S (the same for H^s) is: $w((C, (\alpha_i))) = F^s_b((C, (\alpha_i))) = \sum \alpha_i F^s_b(c_i)$. When $w(C, (\alpha_i)) \geq 0$, we say that $(C, (\alpha_i))$ is a* ***positive h-cycle*** *of H^s.*

As an example, the set $C = \{(x_1 + x_0 - x_2 - x_3 \leq 3), (x_2 + x_0 - x_1 - x_4 \leq -4), (x_4 + x_3 - x_0 - x_0 \leq 5)\}$ generates a positive h-cycle as the sum of these constraints is equal to $3 - 4 + 5 = 4$. Next, we present some results of positive h-cycles.

Theorem 7. *Assume that all hypercycles of H^s are positives and let's take $c \in 4 - \Phi(X^0)$ such that $F_v(c) \neq e_0$. Then, for each h-path P of c, we can find a simple h-path Q of c with a weight less than P.* □

Intuitively, the theorem establishes that we have to consider only simple h-paths when searching for the minimum weight of a hyperarc (nodes of the hyperarc).

Proof. Let $(P, (\lambda_i)_{i\in[1,k]}) \in HPath(c)$ such that $P = \{p_1, p_2, \cdots, p_k\}$, and

$$F_v(\overline{c}) + \sum_{i=1}^{k} \lambda_i F_v(p_i) = \boldsymbol{e_0} \tag{4.4}$$

Recall that, by definition, normal vectors of P are all distinct. If $k = 1$ then P is simple. If P is simple then $Q = P$. Now, assume that P is not simple. Then, we can find a subset $P1 = \{q_1, q_2, \cdots, q_r\}$ (at most with k elements) of $P \cup \{\overline{c}\}$ having the size r, such that the corresponding normal vectors are positively dependent and thus generates a h-cycle (remember that $P \cup \{\overline{c}\}$ generates a h-cycle). We identify two cases: either all $P1$ include $\overline{c}$ ($\overline{c} \in P1$) or there exists $P1$ such that $\overline{c} \notin P1$.

1. Case 1: $\overline{c} \notin P1$. Without loss of generality, assume that $P1 = \{p_1, p_2, \cdots, p_r\}$ such that,

$$\sum_{i=1}^{r} \alpha_i F_v(p_i) = \boldsymbol{e_0} \tag{4.5}$$

   As all hypercycles are positive, then:

$$\sum_{i=1}^{r} \alpha_i F_b^s(p_i) \geq 0 \tag{4.6}$$

   Let $j \leq r$ such that $m_j = \frac{\lambda_j}{\alpha_j} = min(\{\frac{\lambda_i}{\alpha_i} \mid i \in [1, r]\})$. Note that $\frac{\lambda_i}{\alpha_i} - m_j \geq 0$, and $\frac{\lambda_j}{\alpha_j} - m_j = 0$. As,

$$\sum_{i=1}^{k} \lambda_i F_v(p_i) = \sum_{i=1}^{r} \lambda_i F_v(p_i) + \sum_{i=r+1}^{k} \lambda_i F_v(p_i) = \sum_{i=1}^{r} m_j \times \alpha_i F_v(p_i) + \sum_{i=1}^{r} (\lambda_i - m_j \times \alpha_i) F_v(p_i) + \sum_{i=r+1}^{k} \lambda_i F_v(p_i) \tag{4.7}$$

$$F_b^s((P,(\lambda_i))) = \sum_{i=1}^{k} \lambda_i F_v(p_i) = m_j \times \sum_{i=1}^{r} \alpha_i F_v(p_i) + \sum_{i=1}^{r} \alpha_i \times (\frac{\lambda_i}{\alpha_i} - m_j) F_v(p_i) + \sum_{i=r+1}^{k} \lambda_i F_v(p_i) \tag{4.8}$$

From equations (4.8), (4.5) and (4.4), we deduce that

$$\boldsymbol{e_0} = F_v(\overline{c}) + \sum_{i=1}^{k} \lambda_i F_v(p_i) = F_v(\overline{c}) + \sum_{i=1}^{r} \alpha_i \times (\frac{\lambda_i}{\alpha_i} - m_j) F_v(p_i) + \sum_{i=r+1}^{k} \lambda_i F_v(p_i) \tag{4.9}$$

In other words, we have constructed a new h-path Q such that $Q \cup \{\overline{c}\}$ is a h-cycle with at most k elements as $\alpha_j(\frac{\lambda_j}{\alpha_j} - m_j) = 0$. Now, from equations (4.8) and (4.6), we deduce that Q has a weight less than P:

$$\sum_{i=1}^{r} \alpha_i \times (\frac{\lambda_i}{\alpha_i} - m_j) F_b^s(p_i) + \sum_{i=r+1}^{k} \lambda_i F_b^s(p_i) \leq m_j \times \sum_{i=1}^{r} \alpha_i F_b^s(p_i) + \sum_{i=1}^{r} \alpha_i \times (\frac{\lambda_i}{\alpha_i} - m_j)$$
$$F_b^s(p_i) + \sum_{i=r+1}^{k} \lambda_i F_b^s(p_i) \tag{4.10}$$

More specifically, we define $Q = (q_1, q_2, \cdots, q_k)$ and (β_i) by:

- For $i \leq r$, then
  (a) if $\lambda_i - m_j \times \alpha_i \neq 0$ then $q_i = p_i$ and $\beta_i = \alpha_i \times (\frac{\lambda_i}{\alpha_i} - m_j)$
  (b) else drop q_i from Q (we drop elements of P that are dependent in P).
- For $r+1 \leq i \leq k$, then $q_i = p_i$ and $\beta_i = \lambda_i$.

From equations (4.9) and (4.10), it is easy to see that Q is a h-path of c such that: $F_b^s((Q,\beta_i)) \leq F_b^s((P,\lambda_i))$.

Note that Q has at least one element. In fact,

(a) if $r = k$, then there exists at least one index i such that $\lambda_i - m_j \times \alpha_i \neq 0$ otherwise $F_v(\overline{c}) = \boldsymbol{e_0}$.

(b) if $r < k$, at least Q has $k - r \geq 1$ elements.

In this way, we have constructed a h-path having at least one element with less weight than P. If Q is not simple, we replace P with Q and repeat this reasoning until having a simple h-path of c.

2. Case 2: $\overline{c} \in P1$. In this case, all h-cycles of $P \cup \{\overline{c}\}$ contain $\overline{c}$. Let us show that P is the sum of at least two h-paths of c. Without loss of generality, we assume that $P1 = \{\overline{c}, p_1, p_2, \cdots, p_r\}$, with

$$F_v(\overline{c}) + \sum_{i=1}^{r} \alpha_i F_v(p_i) = \boldsymbol{e_0} \tag{4.11}$$

Note that $r < k$ as $P1 \subset P \cup \{\overline{c}\}$. In the same way, we set $j \leq r$ such that $m_j = \frac{\lambda_j}{\alpha_j} = min(\{\frac{\lambda_i}{\alpha_i} \,|\, i \in [1, r]\})$. First, let us show that $m_j < 1$. In fact, if $m_j \geq 1$, then from equation (4.8), we deduce that:

$$\sum_{i=1}^{k} \lambda_i F_v(p_i) = \sum_{i=1}^{r} \alpha_i F_v(p_i) + (m_j - 1) \times \sum_{i=1}^{r} \alpha_i F_v(p_i) + \sum_{i=1}^{r} \alpha_i \times (\frac{\lambda_i}{\alpha_i} - m_j) F_v(p_i) + \sum_{i=r+1}^{k} \lambda_i F_v(p_i) \tag{4.12}$$

From equations (4.4), (4.11), and by adding $F_v(\overline{c})$ to both sides of equation (4.12), we deduce

$$\boldsymbol{e_0} = (m_j - 1) \times \sum_{i=1}^{r} \alpha_i F_v(p_i) + \sum_{i=1}^{r} \alpha_i \times (\frac{\lambda_i}{\alpha_i} - m_j) F_v(p_i) + \sum_{i=r+1}^{k} \lambda_i F_v(p_i) \tag{4.13}$$

In other words, we found a h-cycle of $P \cup \{\overline{c}\}$ not containing $\overline{c}$ (contradiction). Thus, $m < 1$. Now, let us prove that P is the sum of two h-paths of c. Let us add $F_v(\overline{c})$ to both sides of equation (4.8) :

$$F_v(\overline{c}) + \sum_{i=1}^{k} \lambda_i F_v(p_i) = (1 - m_j) F_v(\overline{c}) + m_j F_v(\overline{c}) + m_j \times \sum_{i=1}^{r} \alpha_i F_v(p_i) + \sum_{i=1}^{r} \alpha_i \times (\frac{\lambda_i}{\alpha_i} - m_j) F_v(p_i) + \sum_{i=r+1}^{k} \lambda_i F_v(p_i) \tag{4.14}$$

Again, by reducing equation (4.14) using equations (4.4) and (4.11), we have:

$$\boldsymbol{e_0} = (1 - m_j)F_v(\bar{c}) + \sum_{i=1}^{r} \alpha_i \times (\frac{\lambda_i}{\alpha_i} - m_j)F_v(p_i) + \sum_{i=r+1}^{k} \lambda_i F_v(p_i) \tag{4.15}$$

This is nothing more than a new h-path $Q1$ of c having the weight

$$F_b^s((Q1, (\beta_i)) = \frac{1}{1 - m_j} \times (\sum_{i=1}^{r} \alpha_i \times (\frac{\lambda_i}{\alpha_i} - m_j)F_b^s(p_i) + \sum_{i=r+1}^{k} \lambda_i F_b^s(p_i)) \tag{4.16}$$

In the same way, equation (4.11) defines a h-path $Q2$ of c such that:

$$F_b^s((Q2, (\alpha_i)) = \sum_{i=1}^{r} \alpha_i F_b^s(p_i) \tag{4.17}$$

By replacing equations (4.17) and (4.16) in equation (4.8) and taking the weight function, we have:

$$F_b^s((P, (\lambda_i))) = m_j \times F_b^s((Q2, (\alpha_i)) + (1 - m_j) \times F_b^s((Q1, (\beta_i)) \tag{4.18}$$

Thus, the weight of P is written as an affine combination of the weights of $Q1$ and $Q2$. Thus, one of them has less weight than P. Again, we found a h-path of c with less weight than P.

At the end, we showed that, in all cases, we can find a simple h-path having less weight than P. □

□

As proved in the previous theorem, simple h-paths play an outstanding role in weighted hypergraph. In the next theorem, we establish their uniqueness.

Theorem 8. *Let's assume that $P = (p_1, p_2, \cdots, p_k)$ generates in H^s a simple h-path of $c \in 4 - \Phi(X^0)$ such that $F_v(c) \neq e_0$. Then, there exists a unique family, noted $U(P)$, of scalars $(\lambda_i)_{i\in[1,k]}$ such that $(P, (\lambda_i)_{i\in[1,k]}) \in HPath(c)$.* □

Proof. Let $(P, (\frac{\lambda_i}{\lambda})_{i\in[1,k]}) \in HPath(c)$ and $(P, (\frac{\alpha_i}{\alpha})_{i\in[1,k]}) \in HPath(c)$ two simple h-paths of

c. Thus

$$\lambda F_v(\overline{c}) + \sum_{i=1}^{k} \lambda_i F_v(p_i) = \alpha F_v(\overline{c}) + \sum_{i=1}^{k} \alpha_i F_v(p_i) = \boldsymbol{e_o}$$

As the family $f = \{(F_v(p_i))_{i\in[1,k]} \cup \{F_v(\overline{c})\}$ is simple positively dependent, according to theorem 6, there exists a unique solution $U(f) = ((\beta_i)_{i\in[0,k]})$. Thus, there exists $m \in \mathbb{N}^{>0}$ such that $\lambda_i = m \times \beta_i$, $\alpha_i = m \times \beta_i$, $\lambda = m \times \beta_0$ and $\alpha = m \times \beta_0$. Hence, $\frac{\lambda_i}{\lambda} = \frac{\alpha_i}{\alpha} = \frac{\beta_i}{\beta_0}$. At the end, the unique solution is $U(P) = ((\frac{\beta_i}{\beta_0})_{i\in[1,k]})$. □ □

Theorem 9. *All hypercycles of H^s are positive if and only if all simple hypercycles of H^s are positive.* □

Proof. The first implication is trivial since simple hypercycles are hypercycles. Now, let $(C, (\lambda)_{i\in[1,k]})$ be a hypercycle such that $C = \{c_1, c_2, \cdots, c_k\}$ and let us prove that $\sum_{i=1}^{k} \lambda_i F_b^s(c_i) \geq 0$. Note that $k > 1$ (a h-cycle has at least two elements), and $\sum_{i=1}^{k} \lambda_i F_v(c_i) = \boldsymbol{e_0}$ implies that $P = \{c_2, \cdots, c_k\}$ generates a h-path of $\overline{c_1}$ with weight $\sum_{i=2}^{k} \frac{\lambda_i}{\lambda_1} F_b^s(c_i)$. According to theorem 7, we can find a simple h-path $(Q, (\frac{\alpha_i}{\alpha})_{i\in[1,r]})$ of $\overline{c_1}$ such that $\sum_{i=1}^{r} \frac{\alpha_i}{\alpha} F_b^s(q_i) \leq \sum_{i=2}^{k} \frac{\lambda_i}{\lambda_1} F_b^s(c_i)$. As $Q \cup \{\overline{c_1}\}$ is a simple h-cycle then $\sum_{i=1}^{r} \frac{\alpha_i}{\alpha} F_b^s(q_i) + F_b^s(c_1) \geq 0$, and thus $\lambda_1 \times (\sum_{i=2}^{k} \frac{\lambda_i}{\lambda_1} F_b^s(c_i) + F_b^s(c_1)) \geq 0$. □ □

4.4.3 Minimum weight hypergraph

Generally, canonicity of the systems of linear inequalities is a key point when dealing with CSPs. However, as stated in [39] (regarding roughly similar constraints i.e. octahedra constraints), computing the canonicity is hard: "finding an efficient algorithm that can compute the canonical form of an octahedron from a non-canonical system of inequalities is an open problem at the time of writing this part ". In this part, we will introduce, for the first time, a graph-based characterization for the 4-CSP canonical form, which might lead to the development of new efficient algorithms for other CSP classes.

Definition 20. *Let S be a 4-CSP, and $H^s = (N, E, w)$ be the weighted hypergraph associated to S. The minimum weight hypergraph associated to S is the weighted hypergraph defined by $H_m^s = (N, E, w_m)$, where w_m is the weight function defined on C_s and derived from F_{bm}^s as follows:*

$$w_m(c) = \begin{cases} F^s_{bm}(c) & \text{if } c \in C_s \\ +\infty & \text{else} \end{cases}$$

Whereas F^s_{bm} is defined on the set $4 - \Phi(X^0)$ as follows:

$$\begin{array}{rccl} F^s_{bm}: & 4-\Phi(X^0) & \longrightarrow & \overline{\mathbb{T}} \\ & c & \longmapsto & min(\{\, F^s_b((P,(\lambda_k))) \mid (P,(\lambda_k)) \in HPath(c)\}) \end{array}$$

□

Since the upper bound of c in S might not be a tight upper bound, the minimum weight function F^s_{bm} searches for the tight upper bound of c, if it exists, by taking the smallest bound of all h-paths of c.

Theorem 10. *The following assertions are equivalent:*

1. *All simple hypercycles of H^s are positive*
2. *All simple hypercycles of H^s_m are positive.* □

This theorem states that the minimum weight function preserves the positivity of hypercycles in H^s and H^s_m.

Proof. Let $C = \{c_1, c_2, \cdots, c_r\}$ be a simple h-cycle such that $\sum_{k=1}^{r} \lambda_k F_v(c_k) = \boldsymbol{e_0}$

1. Let us assume that every simple h-cycle of H^s is positive and prove that the associated h-cycle to C is positive in H^s_m.
   (a) First, let us prove that for every constraint c_k of C, $F^s_{bm}(c_k) \neq -\infty$.
      i. According to theorem 7, only simple h-paths of c_k can have less weight than c_k.
      ii. According to theorem 8, each simple h-path has a unique solution.
      iii. The number of combinations (subset) that we can construct from the set 4-$\Phi(X^0)$ is finite (at most 2^{n^4})

   Thus, the set of simple h-paths of c_k is finite. In other words, either $F^s_{bm}(c_k) = +\infty$ or there exists a simple h-path $P_k = (p^k_1, p^k_2, \cdots)$ of c_k such that $F^s_{bm}(c_k) = \cup(P_k) = F^s_b((P_k, (\lambda^k_i))) = \sum_{i=1} \lambda^k_i F^s_b(p^k_i)$.

(b) Now, for each c_k of C, the minimal h-path of c_k will be denoted by P_k. As $\lambda_k \times (F_v(\overline{c_k}) + \sum_{i=1} \lambda_i^k F_v(p_i^k)) = \boldsymbol{e_0}$, thus $\sum_{k=1}^r \lambda_k F_v(\overline{c_k}) + \sum_{k=1}^r \sum_i \lambda_k \lambda_i^k F_v(p_i^k) = \boldsymbol{e_0}$. On the one hand, we know that $C = (c_i, c_2, \cdots, c_r)$ is a simple h-cycle such that $\sum_{k=1}^r \lambda_k F_v(c_k) = \boldsymbol{e_0}$, and we deduce that $\sum_{k=1}^r \lambda_k F_v(\overline{c_k}) = \boldsymbol{e_0}$. On the other hand, $\bigcup_k P_k$ forms a h-cycle and thus $\sum_{k=1}^r F_b^s((P_k, (\lambda_i^k))) \geq 0$ (if simple h-cycles are positive then h-cycles are also positives, from theorem 9). Finally, $\sum_{k=1}^r F_b^s((P_k, (\lambda_i^k))) = \sum_{k=1}^r \sum_i \lambda_k \lambda_i^k F_b^s(p_i^k) = \sum_{k=1}^r \lambda_k F_{bm}^s(c_k) \geq 0$. Thus, C is positives in H_m^s.

2. Let's assume that every simple h-cycle of H_m^s is positive and let's prove that the h-cycle associated to C is positive in H^s. It is easy to see that the path $\{c_2, \cdots, c_r\}$ is a simple h-path of $\overline{c_1}$, and thus $F_{bm}^s(\overline{c_1}) \leq \sum_{k=2}^r \frac{\lambda_k}{\lambda_1} F_b^s(c_k)$. As $F_{bm}^s(c_1) \leq F_b^s(c_1)$, we conclude that $0 \leq F_{bm}^s(c_1) + F_{bm}^s(\overline{c_1}) \leq \frac{1}{\lambda_1} \sum_{k=1}^r \lambda_k F_b^s(c_k)$ (c_1 and $\overline{c_1}$ form a simple h-cycle in H_m^s). □.

□

Next, we will give the fundamental theorem of the feasibility testing of 4-CSP.

Theorem 11. *Let assume that S is bounded. Then $D_s \neq \emptyset$ if and only if all simple hypercycles of H_m^s are positive, where D_s is the solution domain of S.* □

Proof. (sketch).

Without loss of generality, we suppose that S is saturated, which means the existence of all constraints, and if no c_i exists, we should add it (in the way that its minimal bound is infinity). Then, H^s will be complete, and consequently F_b^s of any constraints is bounded. Assume that all simple hypercycles of H_m^s are positive, and let us find a solution to S. The idea of this proof is to use the minimal function to compute minimal bounds and reduce S by adding new constraints until finding a final solution. In fact, starting with $i = 1$, and let us find all constraints c_k such that $\{c_k, c_{i000}\}$ forms a simple h-cycle with $\lambda_k F_v(c_k) + \alpha_k F_v(c_{i000}) = \boldsymbol{e_0}$. Then, we construct a new 4-CSP $S1$ from S, by replacing c_k with c'_k defined by: $F_v(c'_k) = F_v(c_k)$ and $F_b^{s1}(c'_k) = -\frac{\alpha_k}{\lambda_k} F_{mb}^s(c_{i000})$. In other words, we try to find the valuation $\nu \in D_s$ such that $\nu_i = F_{mb}^s(c_{i000})$. Now, S_1 defines the hypergraph H^{s_1}. Note that, the differences

between H^s and H^{s_1}, are only over the weight of hyperarcs (constraints) c_k. We can affirm then that all h-cycles of H^{s_1} are positive. In fact, if we find a h-cycle which is negative, it must necessarily contain some modified constraints c_k: $\sum_{i=1}^{r} \lambda_i F_b^{s1}(c_i) < 0$. This is not possible because in that case we will find a new path of c_{1000} strictly less than $F_{mb}^s(c_{i000})$ (H_m^s is minimal). According to theorem 9, all h-cycles of H^{s_1} are positive implies that all simple h-cycles of H^{s_1} are positive. According to theorem 10, all simple h-cycles of $H_m^{s_1}$ will be positive. Now, given $S1$ and $H_m^{s_1}$, we restart the next iteration $i = 2$, . After at most n iterations, D_s will be reduced to one valuation that satisfies S. □

At the end, the minimum weight hypergraph of S is saturated in the sense that all bounds are reachable.

Theorem 12. *If $D_s \neq \emptyset$, then:*

1. *For all (i, j, p, q), if $F_{mb}^s(c_{jipq}) \neq +\infty$, then there exists $\nu \in D_s$ such that $(\nu_i - \nu_j) - (\nu_p - \nu_q) = F_{mb}^s(c_{jipq})$.*

2. *For all (i, j, p, q), if $F_{mb}^s(c_{jipq}) = +\infty$, then for all $M < +\infty$, there exists $\nu \in D_s$ such that $(\nu_i - \nu_j) - (\nu_p - \nu_q) \geq M$.* □

Proof. (similar to the previous proof) □

4.4.4 Computation method

As stated in the introduction, solving a given CSP aims to achieve one or more goals. In the case of our 4-CSP, we aim to:

- Detect an inconsistency.
- Guarantee the existence of at least one solution.
- Reduce all interval domains to smaller sizes.
- Achieve a solved (or canonical) form wherefrom all solutions can be generated easily.

As will be detailed in this section, these goals can be achieved using the hypergraph-based characterization introduced in the previous section.

Computing the canonical form of a 4-CSP, using the minimal weight function, will provide a useful mechanism to solve many problems modeled by 4-CSP. However, finding an efficient algorithm that can compute the minimal weight function is, in the general case, an open problem at the time of writing this part. Note that, computing the minimal weight by finding all $HPath$, is a hard problem since there are exponential number of $HPath$. Thus, as long as an upper approximation can be guaranteed, an exact representation of a 4-CSP is not required. Keeping this fact in mind, we introduce some fundamental results that will allow us to compute either the canonical form (for some special cases), or an upper approximations of the canonical form. The next theorem gives the necessary conditions to be verified by the minimal weight function.

Theorem 13. *Let $x_i, x_j, x_p, x_q, x_k, x_l$ be six variables of X^0 and let M_{ijpq} denotes the minimal bound $F^s_{bm}(c_{ijpq})$ of a constraint $c_{ijpq} = ((x_i - x_j) - (x_p - x_q) \leq m_{ijpq})$. Then,*

1. $M_{ijpq} = M_{qpji} = M_{ipjq}$
2. $M_{ijkk} = M_{ij00}$
3. $M_{ijji} = 2M_{ij00}$
4. $M_{ijpq} \leq M_{ijkl} + M_{klpq}$
5. $M_{ijpq} \leq M_{iklq} + M_{kjpl}$ □

Proof. The proof of the first point is based on the fact that $F_v(c_{ijpq}) = F_v(c_{ipjq}) = F_v(c_{qpji})$ and thus $HPath(c_{ijpq}) = HPath(c_{ipjq}) = HPath(c_{qpji})$. The same remark holds for points 2 and 3: $F_v(c_{ijkk}) = F_v(c_{ij00})$ and $F_v(c_{ijji}) = 2F_v(c_{ij00})$.
Now, the idea of proving the 4 point comes from the fact that M_{ijpq} is either $-\infty$ (presence of negative hypercycles in S) or reached by a h-path. Here we give only the proof for the first point; the last one can be proved similarly.

- Let assume that $M_{ijkl} \neq -\infty$ and $M_{klpq} \neq -\infty$. Then, there exist two h-paths $(P1, (\lambda_i)) \in HPath(c_{ijkl})$ and $(P2, (\lambda'_i)) \in HPath(c_{klpq})$ such that:

  1. $P1 = \{c_1, c_2, \cdots\}$, $M_{ijkl} = F_b^s((P1, (\lambda_i))) = \sum \lambda_i F_v(c_i)$, and $F_v(\overline{c_{ijkl}}) + \sum \lambda_i F_v(c_i) = \boldsymbol{e_0}$
  2. $P2 = \{c'_1, c'_2, \cdots\}$, $M_{klpq} = F_b^s((P2, (\lambda'_i))) = \sum \lambda'_i F_v(c'_i)$, and $F_v(\overline{c_{klpq}}) + \sum \lambda'_i F_v(c'_i) = \boldsymbol{e_0}$.

  Since $F_v(\overline{c_{ijkl}}) + F_v(\overline{c_{klpq}}) = F_v(\overline{c_{ijpq}})$, and $F_v(\overline{c_{ijpq}}) + \sum \lambda_i F_b^s(c_i) + \sum \lambda'_i F_b^s(c'_i) = \boldsymbol{e_0}$, $P_1 \cup P_2$ generates a h-path of c_{ijpq}. Then, M_{ijpq} is less than the h-path bound associated to $P_1 \cup P_2$ which has as bound of $\sum \lambda_i F_b^s(c_i) + \sum \lambda'_i F_b^s(c'_i) = M_{ijkl} + M_{klpq}$ and thus $M_{ijpq} \leq M_{ijkl} + M_{klpq}$.

- Now, assume that $M_{ijkl} = -\infty$. As every h-path of c_{ijkl}, on the one side, can be extended to a h-path of c_{ijpq}, and on the other side, has a new h-path smaller than it ($M_{ijkl} = -\infty$), then $M_{ijpq} = -\infty$. A similar proof remains valid if $M_{klpq} = -\infty$ □

□

The next theorem, presented below, deals with 4-CSP subclasses solutions.

Theorem 14. *The way we can get the canonical form is given for some subclasses of 4-CSP as follows:*

1. ***Octagon forms**: if all 4-constraints are of the form $(\pm x_i \pm x_j \leq k)$, then the canonical form is given by the first four points of theorem 13.*
2. ***Upper bound forms**: if all 4-constraints are of the form $(x_i - x_j \leq x_p + k)$, then the canonical form is given by the five points of theorem 13.*
3. ***Lower bound forms**: if all 4-constraints are of the form $(x_p \leq x_i - x_j + k)$, then the canonical form is given by the five points of theorem 13.*

Proof. The different subclasses of 4-CSP are based on the nature of their constraints, which result in the three following subclasses:

1. Octagon subclass:

   Foremost, the octagon inequalities are translated into the 4-CSP atomic constraints as follows: $x_i - x_j - x_0 + x_0 \leq M_{ij00}$, $x_0 - x_i - x_j + x_0 \leq M_{0ij0}$, and so on. The initial constraints are then: $c_{i000}, c_{ij00}, c_{0ij0}, c_{00ij}$. If we take into account the four first points of theorem 13, all other constraints can be derived from these initial ones. c_{ijpq} can be obtained, for instance, from c_{ij00} and c_{00pq}.
   The challenge is to prove that if all the minimal bounds of the octagon verify the first four points of theorem 13, then the octagon is surely in its canonical form. Formally speaking:

$$\forall (x_i, x_j) \in \mathbb{R}^2, \pm x_i \pm x_j \leq k \implies \nexists k' \in \mathbb{R} \text{ such that: } \pm x_i \pm x_j \leq k' \leq k.$$

   This assertion will be proved by contraposition. Suppose that the octagon is not canonic even when the four points of theorem 13 are verified, then there exists, for instance, a hyperpath $P = (p_1, p_2, \ldots, p_n)$ such that: $F_v(P) = F_v(c_{ij00})$ and $F^s_{mb}(P) < M_{ij00}$. $P \cup c_{ij00}$ generates a h-cycle, which means: $F_v(c_{ji00}) + \Sigma_{i=1}^{n} \lambda_i F_v(p_i) = e_0$.

   Suppose that this hyperpath length equals to one, i.e. $P = (p_1, p_2)$ then: $\exists M_{i000}$ and M_{0j00} such that: $M_{i000} + M_{0j00} < M_{ij00}$. This is absurd since the fourth point of theorem 13 is already fulfilled. Now, for any hyperpath length, i.e. $P = (p_1, p_2, \ldots, p_n)$ then $\exists M_{c_1}, M_{c_2}, ..., M_{c_n}$ such that: $\sum M_{c_i} < M_{ij00}$. Each constraint c_i is either in or derived from the initial form. Let's replace all constraints by their initial ones, for example: c_{ijpq} can be replaced by two constraints having the lowest bounds (c_{ij00} and c_{00pq} for instance). By doing this we can deduce by induction that: $\exists M_{c_n}$ and M_{c_m} such that: $M_{c_n} + M_{c_m} < M_{ij00}$ and this contradicts the fourth point of theorem 13.

2. The upper bound subclass:

   This form contains three variables per inequality. Considering the proof of the octagon case and Theorem 13, the need for using the first four points of the Theorem 13 to get the canonical form can be proved easily. Let us prove the necessity of the fifth point: $M_{ijp0} \leq M_{ikl0} + M_{kjpl}$:

Let assume that $M_{ikl0} \neq -\infty$ and $M_{kjpl} \neq -\infty$. Then, there exists two h-paths $(P1, (\lambda_i)) \in HPath(c_{ikl0})$ and $(P2, (\lambda'_i)) \in HPath(c_{kjpl})$ such that:

(a) $P1 = \{c_1, c_2, \cdots\}$, $M_{ikl0} = F_b^s((P1, (\lambda_i))) = \sum \lambda_i F_v(c_i)$, and $F_v(\overline{c_{ikl0}}) + \sum \lambda_i F_v(c_i) = \boldsymbol{e_0}$

(b) $P2 = \{c'_1, c'_2, \cdots\}$, $M_{kjpl} = F_b^s((P2, (\lambda'_i))) = \sum \lambda'_i F_v(c'_i)$, and $F_v(\overline{c_{kjpl}}) + \sum \lambda'_i F_v(c'_i) = \boldsymbol{e_0}$.

As $F_v(\overline{c_{ikl0}}) + F_v(\overline{c_{kjpl}}) = F_v(\overline{c_{ijp0}})$, and $F_v(\overline{c_{ijp0}}) + \sum \lambda_i F_b^s(c_i) + \sum \lambda'_i F_b^s(c'_i) = \boldsymbol{e_0}$, thus $P_1 \cup P_2$ generates a h-path of c_{ijp0}. Finally, M_{ijp0} is less than the h-path bound associated to $P_1 \cup P_2$ which has as bound of $\sum \lambda_i F_b^s(c_i) + \sum \lambda'_i F_b^s(c'_i) = M_{ikl0} + M_{kjpl}$ and thus $M_{ijp0} \leq M_{ikl0} + M_{kjpl}$, which is a special case for the fifth property: $M_{ijpq} \leq M_{iklq} + M_{kjpl}$.

3. The lower bound subclass:

   This form contains also three variables per inequality. The result is proved in the same manner as upper bound forms taking into account just the order matter.□

□

4.5 Implementation

After presenting all necessary ingredients and theoretical backgrounds related to the 4-constraint satisfaction problem, we discuss in this section the implementation of a 4-CSP. We answer, from an implementation point of view, to the question of how 4-CSP can be stored and how efficient algorithms can be developed for computing canonical forms and testing the emptiness of a 4-CSP.

4.5.1 2D-DBM data-structure

Difference Bound Matrix (DBM) is a square matrix M where each coordinate m_{kl} represents the upper bound of the difference $x_l - x_k$. For example, the following constraints $x_1 \leq 4$

(equivalent to $x_1 - x_0 \leq 4$), $x_2 \leq 3, x_2 \geq 5$, $8 \geq x_2 - x_1 \geq 6$ can be represented by the following DBM:

$$M = \begin{array}{c} \\ x_0 \\ x_1 \\ x_2 \end{array} \begin{array}{c} \begin{array}{ccc} x_0 & x_1 & x_2 \end{array} \\ \begin{pmatrix} 0 & 4 & 3 \\ 0 & 0 & 8 \\ -5 & -6 & 0 \end{pmatrix} \end{array}$$

To implement and facilitate the manipulation of the 4-CSP domains, a suitable data structure is needed. Therefore, DBM is extended in two dimensions to obtain the so-called "2D-DBM". A 2-Dimensions Difference Bound Matrix (2D-DBM) is a square matrix M where m_{kl} is the upper bound M_{ijpq} of the constraints C_{ijpq}, for $1 \leq k, l \leq (n+1)^2$: lines and columns become difference of variables instead of variables, as depicted in Figure 4.2.

$$M = \begin{array}{c} \\ x_0 - x_0 \\ x_0 - x_1 \\ \\ x_p - x_q \\ \\ x_n - x_n \end{array} \begin{array}{c} \begin{array}{ccccccc} x_0 - x_0 & x_0 - x_1 & \dots & x_i - x_j & \dots & x_n - x_n \end{array} \\ \begin{pmatrix} 0 & M_{0100} & \dots & M_{ij00} & \dots & M_{nn00} \\ M_{0001} & M_{0101} & \dots & M_{ij01} & \vdots & M_{nn01} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ M_{00pq} & M_{01pq} & \dots & M_{ijpq} & \vdots & M_{nnpq} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ M_{00nn} & M_{01nn} & \dots & M_{ijnn} & \dots & M_{nnnn} \end{pmatrix} \end{array}$$

Figure 4.2: 2D-BDM data structure.

4.5.2 Canonical form computation algorithm

The idea of computing the canonical form (or, sometimes, just an upper approximation) of a given 2D-DBM is based on Theorems 13 and 14. In fact, Theorem 13 gives the necessary conditions to be fulfilled by any canonical 4-CSP (e.g canonical 2D-DBM), whereas Theorem 14 establishes special cases where some of these conditions are sufficient. From the viewpoint of graph theory, both theories rely on the minimal weight hypergraph (the hypergraph closure) associated to a 2D-DBM.

4.5.2.1 The hypergraph closure

The first algorithm developed in this part, for computing the canonical (or, sometimes, just an upper approximation) form of a given 2D-DBM and testing the emptiness of the solution set, is illustrated in the Algorithm 1. From the viewpoint of graph theory, Algorithm 1 allows to minimize the hypergraph associated to a given 4-CSP and to check the existence of a negative hypercycle.

input : 2D-DBM
output: Canonical 2D-DBM (or upper approximation of the canonical form in the worst case)

do
  foreach *cell* M_{ijpq} *in 2D-DBM representing a 4-CSP constraint* **do**
    $M_{ijpq} := min(M_{ijpq}, M_{ijkl} + M_{klpq}, M_{iklq} + M_{kjpl})$
  end
  Update cells in order to ensure the following equalities:
    $M_{ijpq} := M_{qpji} := M_{ipjq}$
    $M_{ijkk} := M_{ij00}$
    $M_{ijji} := 2M_{ij00}$
while *2D-DBM is not yet stationary*;

Algorithm 1: Skeleton of the hypergraph closure

A solution is guaranteed if the diagonal of the final canonical 2D-DBM does not contain any negative cell (i.e. there is no negative hypercycle in the hypergraph). Thus, we obtain at least one solution: the variable valuations contained in the first column. Note that each iteration of the algorithm presents the constraint propagation technique, since the changing of one constraint upper bound impact the upper bounds of the others. In fact, each iteration strengthens the bounds of each system constraint, which means that it excludes quickly many values from the variables domains. Consequently, the constraint propagation process is accelerated.

4.5.2.2 From the hypergraph closure to the 4-CSP tractability

For the sake of clarity, we presented in Algorithm 1 only the skeleton of the hypergraph closure, without giving technical details about how operations will be implemented or when

input : 2D-DBM
output: Canonical 2D-DBM

Variables i, j, p, q, k, l, s: Integers;
/* In this algorithm $[i]$ denotes the integer value of i and $n+1$ the number of domain variables (including the x_0 variable which is always null), and we note:

- $i = [l/(n+1)]$
- $j = l - [l/(n+1)] * (n+1)$
- $p = [k/(n+1)]$
- $q = k - [k/(n+1)] * (n+1)$ */

M: Table;
$iter := 1$;
do
  for $k = 1$ **to** $pow((n+1), 2)$ **do**
    for $l = 1$ **to** $pow((n+1), 2)$ **do**
      /*The following loop serves to update the matrix in order to verify the two last points of the Theorem 13. */
      for $s = 1$ **to** $pow((n+1), 2)$ **do**
        $M[k,l] := min(M[k,l], M[k,s] + M[s,l],$
        $M[p*(n+1)+i, s] + M[s, q*(n+1)+j],$
        $M[p*(n+1)+q, s] + M[s, i*(n+1)+j],$
        $M[j*(n+1)+q, s] + M[s, i*(n+1)+p],$
        $M[(s/[n+1])*(n+1)+i, q*(n+1)+s-[s/(n+1)]*(n+1)]$
        $+M[j*(n+1)+s/[n+1], (s-[s/(n+1)]*(n+1))*(n+1)+p]);$
      end
      /*The following loop serves to update the matrix in order to verify the three first points of the Theorem 13. */
      for $p = 1$ **to** $pow((n+1), 2)$ **do**
        $M[k,l] := min(M[k,l], M[(p*(n+1)+q, i*(n+1)+j], M[p*(n+1)+i, q*(n+1)+j]);$

        if $([l/(n+1)] = l - [l/(n+1)])$ **then** /*$M_{ijkk} = M_{ij00}$ */
          $M[k,l] := M[k,0];$
        end
        if $([k/(n+1)] = l - [l/(n+1)]$ *and* $k - [k/(n+1)] = [l/(n+1)])$ **then**
          /*$M_{ijji} = 2M_{ij00}$ */
          $M[k,l] := 2 * M[k,0];$
        end
      end
    end
  end
  $iter++;$
while $iter <= pow((n+1), 4)/2;$

Algorithm 2: Canonical form of a 2D-DBM

the algorithm will terminate. Technically, in the implementation, we use two two-dimensional tables, with $(n+1)^2$ columns and $(n+1)^2$ lines. The 2D-DBM is rewritten in a way that: column C_{ij} (resp. line L_{ij}) of 2D-DBM which represents the variable $x_i - x_j$ becomes the column C_{i*n+j} (resp. line L_{i*n+j}).

The algorithm complexity analysis. It is obvious that the canonical form is obtained in at most $(n+1)^4/2$ iterations. This maximum number of iterations is achieved if we suppose that just two difference variables are related pairwise, that way we will have $(n+1)^4/2$ binary classes.

Our algorithm is polynomial in time and space. In fact it has a complexity $\mathcal{O}(n^{10})$. This reflects the efficiency of our algorithm compared with the other approximation algorithms that infer complex constraints, and the precision obtained besides using just binary constraints interested in by the majority of works.

4.5.2.3 The whole algorithm

Up to now, the domain of solutions is very reduced, it remains fair to extract the solution combinations. For this, we add to our algorithm the last version of the Arc Consistency algorithm AC2001 [25]. AC2001 takes as an only input the Constraint Network resulted from the hypergraph closure algorithm, which is very reduced. Therefore, it provides the set of variable values quickly.

input : Constraint Network (N, D, C)
output: Solutions of the CN

1 The hypergraph closure function on 2D-DBM (Algorithm 2)
2 Take bounds of variables (for domains D) and bounds of binary relations from 2D-DBM (for constraints C)
3 Accomplish the arc consistency algorithm AC2001

Algorithm 3: The Whole Algorithm Skeleton

4.6 Conclusion

In this part, we have introduced a subclass of CSP named 4-CSP. As it has been shown, studying 4-CSP can be of great importance, considering their omnipresence in many real problems as well as their reduced complexity proven to be polynomial. In comparison with the other variants of CSP, the 4-CSP is more rich than binary CSP in terms of invariants precision; and less complex than the general CSP in terms of implementation cost, since it is proved to be cubic in the number of system variables. The main contribution of this part consists of providing a complete framework for the 4-CSP, including the theoretical background and the implementation issues.

In addition, we have also provided the first answer, to the best of our knowledge, to the following fundamental problem : can we build a scalable and graph theory based algorithms for CSP tractability similar to those of Bellman? Thanks to the hypergraph theory coupled with positive linear dependence theory, a positive answer has been proved for the 4-CSP class. This result might be extended to CSP with constraints similar to those of the Octahedra [39].

Finally, in order to represent and manipulate 4-CSP, we have defined a suitable data-structure called 2D-DBM, and elaborated the algorithm able to obtain the canonical form for this structure.

Chapter 5

4-CSP applications

5.1 Parametric real-time systems

As affirmed, the verification of Parametric Timed Automata is generally undecidable. However, it is decidable for some restricted subclasses and many practical systems even they are not included in this subclasses can be verified using heuristics. Searching the decidability, we consider the Lower/Bound timed automata where the parameters appears either as lower bound or upper bound. The challenge is to find an efficient data structures to express the constraints of the system. We use the 2D-DBM explained in the previous chapter.

5.1.1 4-constraints in parametric timed automata

We consider the parametric real-time systems, then the constraints are expressed on clocks and parameters. Indeed, the constraints considered are the 4-constraints: $(x-y)-(z-t) \leq C$ such that each of the variables x, y, z and t can be clock or parameter. In the following, we show how our algorithm serves for symbolic analysis of these systems.

Forward analysis with the operation *post()*

As explained in the first chapter of this thesis, the operation *post(H,t)* returns a zone that contains all states that can be reached by $\mathcal{A}$ after it performs transition t from zone H. The

operation *post(H,t)* is defined as follows:

$$post(H,t) = (l', ((Z^{\uparrow} \cap g)[r := 0]) \cap Inv(l')) \tag{5.1}$$

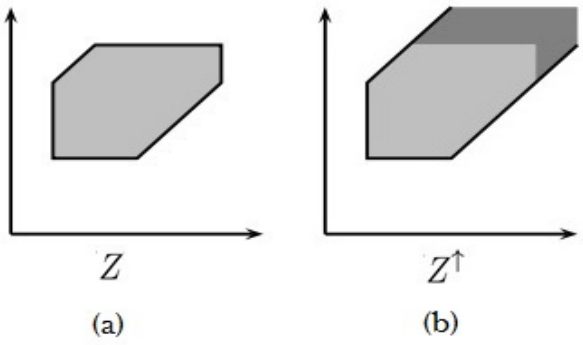

Figure 5.1: The zone and its future

The zone presents the constraints of a symbolic state. In our case, it is presented by the 4-CSP which is presented in its turn by the matrix 2D-DBM 5.1 (a). Then the future of the zone is computed, it is accomplished by incrementing columns representing variables by the same value (b). After that, we update the 2D-DBM by the constraints imposed by the guard 5.2 (a) and we reset variables corresponding to some clocks during transition (b).

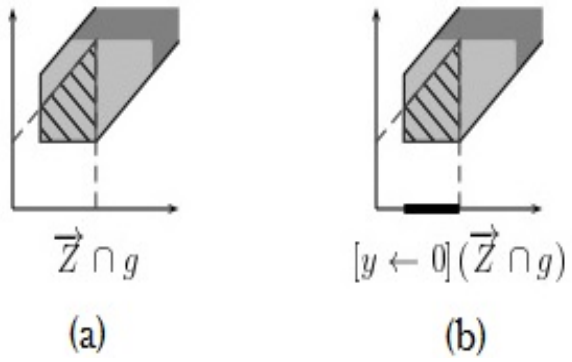

Figure 5.2: Intersection with guard and clock update

The algorithm we developed is applied every time we finish the post () method in order to:

- Determine the accessibility of the following states.
- Detect inconsistencies in invariants and guards.

At the end of the on the fly analysis we can:

- Determine the (ranges of) values of the parameters.
- Check the accessibility of the final states.

5.2 4-Octahedron abstract domain

5.2.1 The need for 4-octahedron abstract domain

The choice of a suitable abstract domain to be use for approximating a given problem, on one hand, has a great impact on the precision of the specification to be proved and reducing false alarms. On the other hand, the complexity of an abstract domain operations may limit its applicability to real problems. Convex polyhdra and octahedra abstract domains are very precise, flexible and expressive domains for static analysis of programs and verification of reactive systems. However, these domains have a high computational complexity (exponential complexity in general). Here, we give two examples of problems that can be efficiently approximated by our 4-Octahedra abstract domain.

Static code analysis. Let us consider the following code fragment:

```
x=rand(5,20);

y=rand(3,80);

z=rand(3,7);

t=rand(90,100);
```

```
while (x ¡= z+3 and y ¡= t+6 )
{
    x=x-1;
    y=y-1;
}
```

In order to prove the correctness of this code fragment, it is not sufficient to ensure the constraints: $5 \leq x \leq 20 \wedge 3 \leq y \leq 80 \wedge 3 \leq z \leq 7 \wedge 90 \leq t \leq 100$, but it is necessary to ensure the relational invariants: $y + x - z - t \leq 3 \wedge y + x - z - t \geq 5$.

Related works

In the literature, we find different commonly used numerical abstract domains as depicted in the Table. 5.1.

Abstract Domain	Invariants	Example	Complexity	Reference
Intervals	$k_1 \leq x_i \leq k_2\,, k_1, k_2 \in \mathbb{R}$	$1 \leq x \leq 4$	$\mathcal{O}(n)$	[41]
Difference Bound Matrices (DBMs)	$k_1 \leq x_i \leq k_2\,, k_1, k_2 \in \mathbb{R}$ $x_i - x_j \leq k\ ,\ k \in \mathbb{R}$	$x \leq 2 \wedge x - y \leq 3$	$\mathcal{O}(n^3)$	[45]
Octagons	$\pm x_i \pm x_j \leq k\ ,\ k \in \mathbb{R}$	$x + y \leq 6 \wedge x - y \leq 2$	$\mathcal{O}(n^3)$ or $\mathcal{O}(n^4)$	[76]
4-Octahedra	$x_i \sim k, x_i - x_j \sim k, (x_i - x_j) - (x_p - x_q) \sim k, \sim \in \{\leq, \geq\}, k \in \mathbb{R}$	$x \leq 5 \wedge z \leq 6 \wedge x - y + z \leq t + 3$	$\mathcal{O}(n^4)$	Our paper [81]
Octahedra	$\sum x_i - \sum x_j \geq k\ ,\ k \in \mathbb{R}$	$x + y - z \leq 7$	$\mathcal{O}(3^n)$	[39]
Convex polyhedra	$\sum c_i.x_i \geq k\, c_i, k \in \mathbb{R}$	$2x + y - 4z \geq 3$	Exp	[54]
Templates	Like convex polyhedra, but the set of coefficients c_i is predetermined		Exp	[92]

Table 5.1: Summary of numerical abstract domains based on inequalities

The lattice of Intervals [41] presents invariants of the form $x \in [c1, c2]$. It was extended to Pentagon domain in [71], with a conjunction of inequalities $x < y$.

Analysis using interval domains is efficient since it requires a linear memory and time in the worst case, but it lacks the precision and limits the expressiveness of the computed invariants. In the other side, convex polyhedra is the most rich representation, since it captures all precise constraints, but this offered precision come with a complexity which is exponential in the number of program variables, therefore it is not practical. Then, the challenge is to make a trade-off between the precision and the low-cost, i.e. try to have a polynomial complexity in the implementation while preserving as much as possible the richness of the expressed properties. For this reason, the weakly relational abstract domains has attracted a lot of interest. Namely, the Difference Bound Matrices [77], extended in [84] with disequality constraints, and the Octagons [76], extended as well with absolute value constraints [101] in order to capture non-convex programs properties. The algorithms developed in this context have been implemented in an efficient program analyzers, such as APRON [63], and ASTRÉE [26].

Contributions of the section

The main contributions of this application:

1. We introduce a new abstract domain encompassing the invariants of the form : $\{ x \sim \alpha, x - y \sim \beta, (x - y) - (z - t) \sim \lambda\}$, such that: x, y, z and t are real variables, α, β and λ are real constants and $\sim \in \{\leq, \geq\}$. The precision of our domain lies between the octagon domain which encodes the relation of the type: $\pm x \pm y \leq k$ for $k \in \mathbb{R}$ and the octahedra domain. Our domain may have a great benefit for verification of Parametric Timed Automata (PTA) [7] since most verification problems are undecidable [19, 9].

2. One of the scopes of this application is to explore how we can characterize the feasibility of polyhedra in general, and 4-octahedra in particular. We try to obtain similar results in graph theory able to characterize the feasibility of polyhedra as those of Bellman.

Based on the 4-CSP, we introduce the 4-Octahedron abstract domain, we prove its consistence with the original concrete domain, and we define the data-structure adequate to its representation.

As for convex polyhedra and octahedra, a 4-octahedron abstracts the set of vectors in $\mathbb{T}$ that satisfy a restricted system of linear inequalities. In our case, inequalities are atomic 4-constraints defined in the last chapter. Then, the 4-octahedron abstract domain is defined as follows:

Definition 21. *A **4-octahedron** O over X, written*

$$O = \bigwedge((x_i - x_j) - (x_p - x_q) \leq m_{ijpq})$$

is the solution of m canonical 4-constraints overs X^0. $\square$

The manipulation of this abstract domain comes to exploit the 4-CSP tractability results. After defining the structure of the 4-CSP, i.e. 2D-DBM, it will be easy to prove the 4-octaherdon fidelity to the original concrete domain using the Galois Connections. This step is necessary in order to prove the consistence of the abstract domain and its correctness.

5.2.2 Galois Connections

As said, the abstraction allows us to easily verify some determined properties satisfaction, since this verification is complex or even impossible in the concrete domain. So it is necessary, firstly, to prove that the Abstract Domain created is consistent, i.e. it is connected to the concrete one by a Galois Connection.

The stake is to define the two functions of abstraction γ and concretization α that verifies the Galois connections. Formally speaking, let $C = (D_C, \sqsubseteq_C)$ and $A = (D_A, \sqsubseteq_A)$ two partially ordered sets, the following equivalence should be :

$$\forall c \in C, \forall a \in A \qquad \alpha(c) \sqsubseteq_A a \Leftrightarrow c \sqsubseteq_C \gamma(a) \tag{5.2}$$

Let us define the set of 2D-DBM valuation domain :

$$D = \{(x_0, x_1, ..., x_n) \in \mathbb{R}^{n+1} \,|\, \forall i, j, p, q,\ (x_i - x_j) - (x_p - x_q) \leq M_{ijpq}\} \tag{5.3}$$

Given a canonical 2D-DBM M^c, we can obtain the set of concrete values framed by M^c by

using this following concretization function:

$$\gamma^{2D-DBM}(M^c) : O \rightarrow \mathcal{P}(\nu \rightarrow \mathbb{R}) \tag{5.4}$$

$$\gamma^{2D-DBM}(M^c) = \{(x_0, x_1, ..., x_n) \in \mathbb{R}^{n+1} | i, j, p, q(x_i - x_j) - (x_p - x_q) \leq M_{ijpq}\} = D(M^c)$$

Where the abstraction function α is defined by: $\alpha(D) = \bigcap\{m \in \mathcal{M}_\perp | D \subseteq \gamma(m)\}$. Note that: $\gamma^{2D-DBM}(M^c) = \varnothing$ if $M^c = \perp$.

5.2.3 Definition of the abstract operators

In order to express precisely the whole dynamic behaviour of a program, we should define the abstract semantics of its primitive operators. Namely, the abstract semantics of:

- Transfer functions, that model assignment and test statements.
- Set-theoretic operators, such as union (resp. intersection) that interprets the disjunction (resp. conjunction) of many code fragment invariants conditions.
- Extrapolation operators, such as widening that computes, over-approximations of the variables set in the semantics of loops and recursive functions.

These abstract operators are defined as manipulations of the associated 2D-DBM. The result of these operations is surely an upper approximations of the exact solutions.

5.2.3.1 Intersection.

The point-wise intersection of two 4-Octahedra represented by their canonical 2D-DBM M^c and N^c is obtained as follows:

$$(M^c \wedge N^c)_{ij} = min((M^c)_{ij}, (N^c)_{ij}), \text{ for } 0 \leq i, j \leq (n+1)^2$$

The resulting matrix is usually not closed, so we are forced to accomplish closure algorithm to obtain the canonical form. In the other hand, the intersection is always exact, which means that:

$$\gamma(M^c \wedge N^c) = \gamma(M^c) \cap \gamma(N^c)$$

5.2.3.2 Union.

The Join (Union) operator computes the least upper bound of two abstract domains. It is used to obtain the abstract element after the join program node by merging the abstract elements before this join node. Thus, the union of two 4-Octahedra domains O_1 and O_2 described by two 2D-DBM M^c and N^c is not defined in an exact way, so we get its over-approximation as follows:

$$(M^c \vee N^c)_{ij} = max((M^c)_{ij}, (N^c)_{ij}), \text{ for all } 0 \leq i, j \leq (n+1)^2$$

Remark 2.

- *The resulted matrix is guaranteed to be closed. But the set of valuations that it represents may not be a 4-Octahedron.*

- *Since the computation is just approximated, we have this inclusion:*

$$\gamma(M^c \vee N^c) \supseteq \gamma(M^c) \cup \gamma(N^c)$$

5.2.3.3 Linear assignment.

We assign to the variable x_k a linear expression over other variables: $x_k \longleftarrow L(x_1, x_2, ..., x_n)$. It reflect the semantic given to the assignment operation in the program.

We perform the assignment in the canonical 2D-DBM (M^c), then we close it applying the previous algorithm. The resulted matrix is canonical and preserves the sens of linear combination.

5.2.3.4 Projection.

The projection operation allows to reduce the variables vector dimension of the 4-Octahedra by 1. It suppose that a all information about a variable x_k are lost.
A projection of our 4-Octahedron that removes a dimension x_k, can be accomplished by removing from its canonical associated 2D-DBM (M^c) all columns and lines which concern the variable x_k. Then we perform our closure algorithm.

5.2.3.5 Widening.

In loops program that involves infinite iteration sequences, the widening operator computes the upper approximation of a fixpoint in bounded number of steps. Which means that: For a sequence of 4-Octahedra $O_0, O_1, ...$, such that O_{i+1} succeeds O_i, if we have $O'_0 = O_0$, $O'_1 = Widen(O'_0, O_1)$, ... then there exist $i \geq 0$ such that $O'_j = O'_i$ for all $j > i$. The chain becomes stationary after a finite i iteration.

$$(M^c \bigtriangledown N^c)_{ij} = \begin{cases} (M^c)_{ij} & \text{if } (N^c)_{ij} \leq (M^c)_{ij}, \\ +\infty & \text{elsewhere.} \end{cases}$$

The concrete domains of the two 4-Octahedra and their widening verify the following inclusion property:

$$\gamma(M^c \bigtriangledown N^c) \supseteq \gamma(M^c) \cup \gamma(N^c)$$

Conclusion

In this section, we provided a new abstract domain: the 4-Octahedra, which has an important practical interest in the code static analysis. It is an Octahedra subclass that infers relations of the form : $\{ x \sim \alpha, x - y \sim \beta, (x - y) - (z - t) \sim \lambda\}$, such that: x, y, z and t are real variables, α, β and λ are real constants and $\sim \in \{\leq, \geq\}$. We proved its consistence with the original concrete domain, i.e. the set of 4-constraints inequalities conjunction solutions. In fact, it presents an over-approximation of this solutions set. In order to represent and manipulate this domain and accomplish the program operators as well, we defined a suitable structure 2D-DBM. Then, we elaborated the algorithm able to obtain the canonical form of this structure. To sum up, in term of invariants precision, the 4-Octahedron is more rich than Octagons, and regarding the implementation cost, it is less complex than the polyhedra, since it is proved to be cubic in the number of system variables.

5.3 Multimodal transport

Multimodal transport is a logistic problem in which a set of goods have to be transported to different places with the combination of at least two modes of transport, without a change of container for the goods. Thus, it consists of using in the same path or trip several modes of transport (truck, car, train, plane ...). This technique has emerged to deal with problems such as pollution, energy consumption and especially for reason to reduce congestion. Several issues arise from this idea, we can cite planning of multimodal transportation tasks [14, 10], modeling the multimodal transportation networks. A lot of algorithms have been elaborated to find the viable shortest path under objectives such as travel time and number of modal transfers [3, 17]. The most important among them remains the calculation of multimodal shortest path.

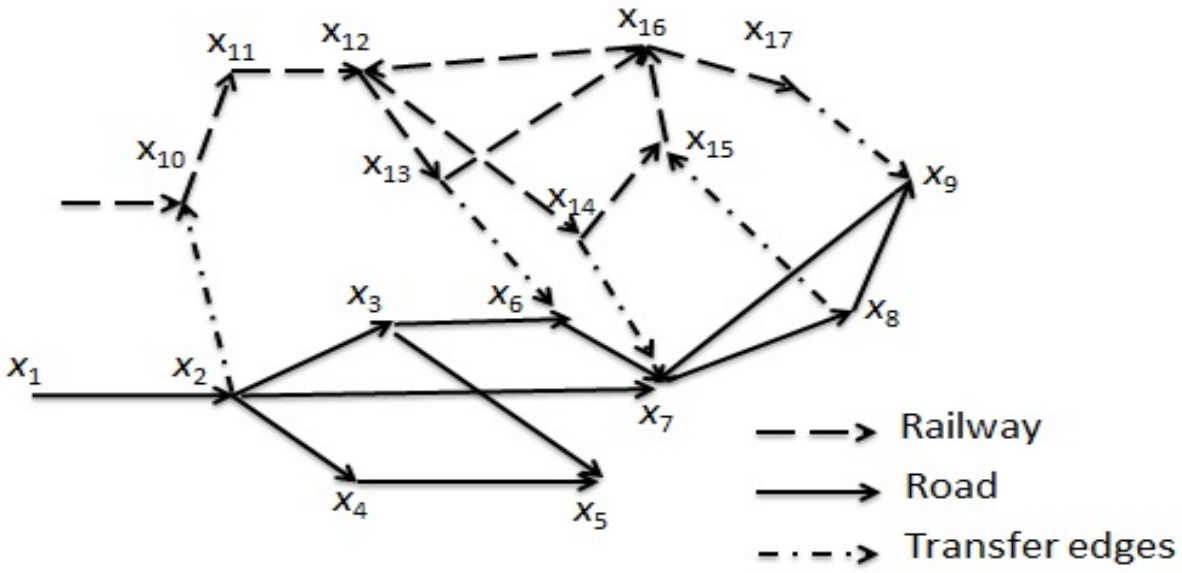

Figure 5.3: Multimodal transport graph

Case of Two Modes

To formulate the problematic, assume that we have a transport network as shown in Figure 5.3. It is presented by a graph GM(V,E) constituted by two modes. The first mode is

the roads (m1) and the second one presents the railways (m2). Each mode has its set of nodes $V_(m_i) \subset V$(its cardinal noted ni) and its edges $E_(m_i) \subset EE$ The graph contains also a set of transfer edges Et that allows the mode change. These edges are mostly directed; therefore we note the set of their heads H and the set of their tails T. During the transportation, we are facing constraints on mode changing, such as minimizing the number of transfer, viability, time of conveyance waiting ...etc. For some reasons such as congestion, we can change the mode only if we will reduce the transport cost. For example, we can choose the path y_1y_4 instead of x_2x_6 path only if the following condition is fulfilled: $Cost(y_1y_4) \leq Cost(x_2x_6)$. In general, we have: $Cost(x_ix_j) \leq Cost(x_kx_l)$, for $(x_i, x_l) \in H^2$ and $(x_j, x_k) \in T^2$.

Given these data and flowing in both modes, the problem is to find the lowest cost between any two points of the graph.

Lowest cost in multimodal transport

We present the transport network by a hypergraph, and the problem of computing the lowest cost as a 4-Constraint Satisfaction Problem.

The constraints over edges of the same mode are expressed as: $x_k - x_l - e_0 - e_0 \leq c$ which means that traveling from x_l to x_k has as maximum cost the value c. On the other hand, the constraints on mode changes are expressed by constraints of the form: $Cost(x_ix_j) \leq Cost(x_px_q)$ which is equivalent to the following constraints in the 4-CSP framework: $x_j - x_i + x_q - x_p \leq 0$. The problem is then comes down to use the algorithm 3.

Case of More Than Two Modes

The previous algorithm provides the shortest path only if the following proposition is considered: between two crossings of the same mode we pass through an odd number of modes. This implies that the start point and the arrival point belong to the same mode. Consequently, to hold the same computation, the use order of modes in the general case should be as following: $m_i, m_j, m_k, m_j, m_k, m_l, m_o, m_p, m_o, m_l, m_k, \ldots, m_i$. In this way, we will have the lowest cost between each two points using the same algorithm.

Conclusion and perspectives

The Constraint Satisfaction Problem (CSP) is a fundamental concept in constraint programming, used fundamentally to model and solve research problems. Most real-world problems can be solved with CSPs, including resource allocation, scheduling, building design, graph coloring, temporal reasoning, maximization of financial profits, and more. path optimization, classification, tomography.

Motivated by several applications in the verification of real-time systems, we have studied in our thesis a class of CSP. It is about "4-Constraint Satisfaction Problem", noted $4-CSP$ which includes the constraints of the form: $\{x \sim \alpha, x-y \sim \beta, (x-y)-(z-t) \sim \lambda\}$, where x, y, z and t are real variables, α, β and λ are real constants and $\sim \in \{\leq, \geq\}$. Compared to other CSP variants, 4-CSP is richer than the binary CSP in terms of invariant precision and less complex than the general CSP in terms of implementation cost, since we have shown that is cubic in number of system variables. We relied on strong mathematical foundations to provide proofs of the 4-CSP tractability. Actually, we elaborated a 4-CSP resolution algorithm based on the closure and the positive linear dependence theories coupled with the constraint propagation technique. The time and space complexities of resolution algorithms are shown to be polynomial and the comparison with other works is discussed.

Once we set up a proven and powerful theoretical basis, we applied the resulted computation techniques and algorithms in many applications such as parametric real-time verification and static code analysis. Mainly, we use this computation power to track the tractability of parametric timing constraints of systems modeled by PTA, providing thereby the forward and backward reachability analysis.

Our work is likely to have interesting extensions. In fact, the idea of mapping constraints to weighted hyperarcs and using the linear independence of vectors to define hyperpaths and hypercycles in the constraint hypergraph along with the use of 2D-DBMs to work with the hypergraphs is an interesting and original idea that may be extended to other CSPs with arity higher than 4. An other strong point is the generalization of cycles in graphs to hypercycles in hypergraphs via the notion of positively dependent vectors. It certainly deserves to be investigated further since it obviously help to compute weight in hypergraphs, and this computation has low complexity.

Moreover, the application areas of our computation methods and algorithms are not restricted to the verification of real-time systems nor static code analysis. Indeed, they can serve in any fields having constraints like those we consider, such as astronomy, logistics, finance, learning, etc.

Bibliography

[1] F. A. Aloul. Search techniques for sat-based boolean optimization. *Journal of the Franklin Institute*, 343(4-5):436–447, Jul 2006.

[2] R. Alur. Timed automata. In Nicolas Halbwachs and Doron Peled, editors, *Computer Aided Verification*, pages 8–22, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[3] R. Alur, C. Courcoubetis, and D. Dill. Model-checking in dense real-time. In *Proc. 5th IEEE Symposium on Logic in Computer Science*, LICS'90, pages 414–425. IEEE Computer Society Press, 1990.

[4] R. Alur, C. Courcoubetis, and D. Dill. Model-checking in dense real-time. *Information and Computation*, 104:2–34, 1993.

[5] R. Alur, C. Courcoubetis, T. A. Henzinger, and P. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*, pages 209–229. Springer.

[6] R. ALUR and D. DILL. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[7] R. Alur, T. A. Henzinger, and M. Y. Vardi. Parametric real-time reasoning. In *Proceedings of 25th Annual Symposium on Theory of Computing*, STOC'93, pages 592–601, New York, USA, 1993. ACM Press.

[8] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183 – 235, 1994.

[9] É. André and N. Markey. Language preservation problems in parametric timed automata. In Vicario E. Sankaranarayanan S., editor, *Proceedings of Formal Modeling and Analysis of Timed Systems*, volume 9268 of *FORMATS'15*, pages 27–43. Springer International Publishing, 2015.

[10] Étienne André. What's decidable about parametric timed automata? *International Journal on Software Tools for Technology Transfer*, pages 1–17, Jul 2017.

[11] K. R. Apt. The essence of constraint propagation. *Theoretical Computer Science*, 221(1-2):179–210, June 1999.

[12] E. Asarin, O. Maler, A. Pnueli, and J. Sifakis. Controller synthesis for timed automata. In *Proceedings of the 5th IFAC Symposium on System Structure and Control*, SSSC'98, pages 469–474. Elsevier Science, 1998.

[13] G. Ausiello, P. G. Franciosa, and D. Frigioni. Directed hypergraphs: Problems, algorithmic results, and a novel decremental approach. *Theoretical Computer Science*, pages 312–328, 2001.

[14] P. Gastin B. Berard, V. Diekert and A. Petit. Characterization of the expressive power of silent transitions in timed automata. *Fundamenta Informaticae*, 36(2-3):145–182, 1998.

[15] C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT-Press, 2008.

[16] R. Barták, Miguel A. Salido, and F. Rossi. New trends in constraint satisfaction, planning, and scheduling: a survey. *Knowledge Engineering Review*, 25(3):249–279, Aug 2010.

[17] F. Basile, P. Chiacchio, and J. Coppola. Identification of time petri net models. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 47(9):2586–2600, Sept 2017.

[18] R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958.

[19] N. Beneš, P. Bezděk, K. G. Larsen, and J. Srba. Language emptiness of continuous-time parametric timed automata. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Proceedings of the International Colloquium on Automata, Languages, and Programming*, volume 9135 of *ICALP'2015*, pages 69–81, Berlin, Heidelberg, 2015. Springer.

[20] J. BENGTSSON, K.G. LARSEN, F. LARSSON, P. PETTERSSON, and C. YI, W.and WEISE. New generation of uppaal. In *International Workshop on Software Tools for Technology Transfer*, volume 1, 1998.

[21] Claude Berge. *Graphes et Hypergraphes*. Dunod Université. Dunod, Paris, 2 edition, 1973.

[22] M. M. Bersani, M. Rossi, and P. San Pietro. A logical characterization of timed regular languages. *Theoretical Computer Sciences*, 658(PA):46–59, January 2017.

[23] B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using time petri nets. *IEEE transactions on software engineering*, 17(3):259–273, 1991.

[24] C. Bessière, P. Meseguer, E. C. Freuder, and Javier Larrosa. On forward checking for non-binary constraint satisfaction. *Artificial Intelligence*, 141(1):205–224, Oct 2002.

[25] C. Bessière, J. C. Régin, R. H. C. Yap, and Y. Zhang. An optimal coarse-grained arc consistency algorithm. *Artificial Intelligence*, 165(2):165–185, Jul 2005.

[26] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. A static analyzer for large safety-critical software. In *Proceedings of Programming Language Design and Implementation*, volume 38 of *PLDI'03*, pages 196–207. ACM Press, May 2003.

[27] I. Blunno, J. Cortadella, A. Kondratyev, L. Lavagno, K. Lwin, and C.P Sotiriou. Handshake protocols for de-synchronization. In *Proceedings of 10th International Symposium on Advanced Research in Asynchronous Circuits and Systems*, ASYNC'2004, pages 149–158. IEEE, April 2004.

[28] M. Bodirsky and M. Pinsker. Schaefer's theorem for graphs. *Journal of the ACM*, 62(19):1–52, Jun 2015.

[29] P. Bouyer. Model-checking timed temporal logics. *Electronic Notes in Theoretical Computer Science*, 231:323 – 341, 2009. Proceedings of the 5th Workshop on Methods for Modalities (M4M5 2007).

[30] P. Bouyer, F. Chevalier, and N. Markey. On the expressiveness of tptl and mtl. In *Proceedings of the 25th Conference on Foundations of Software Technology and Theoretical Computer Sciences*, volume Lecture Notes in Computer Science 3821 of *FST & TCS'05*, pages 432–443, 2005.

[31] P. Bouyer, N. Markey, and O. Sankur. Robust reachability in timed automata and games: A game-based approach. *Theoretical Computer Science*, 563:43 – 74, 2015.

[32] Patricia Bouyer, Serge Haddad, and Pierre-Alain Reynier. Timed petri nets and timed automata: On the discriminating power of zeno sequences. *Information and Computation*, 206(1):73 – 107, 2008.

[33] S. C. Brailsford, C. N. Potts, and Barbara M. Smith. Constraint satisfaction problems: Algorithms and applications. *European Journal of Operational Research*, 119(3):557–581, December 1999.

[34] R. Bruni and U. Montanari. *Discrete Time Markov Chains with Actions and Nondeterminism*, pages 333–341. Springer International Publishing, Cham, 2017.

[35] A. A. Bulatov. A dichotomy theorem for nonuniform csps. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 319–330, Oct 2017.

[36] Daniel Bundala and Joël Ouaknine. Advances in parametric real-time reasoning. In Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik, editors, *Mathematical Foundations of Computer Science (MFCS'14)*, pages 123–134, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[37] D. Chandler. Theory of positive linear dependence. *American Journal of Mathematics*, 76(4):733–746, Oct 1954.

[38] A. Cimatti, L. Palopoli, and Y. Ramadian. Symbolic computation of schedulability regions using parametric timed automata. In *2008 Real-Time Systems Symposium*, pages 80–89, Nov 2008.

[39] R. Clarisò and J. Cortadella. The octahedron abstract domain. *Science of Computer Programming*, 64(1):115–139, Jan 2007.

[40] E. Clarke, E. Emerson, and A. Sistla. Automatic verification of finite state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[41] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM Symposium on Principles of Programming Languages*, POPL'77, pages 238–252. ACM Press, 1977.

[42] P. Cousot and R. Cousot. Abstract interpretation and application to logic programs. *Journal of Logic Programming*, 13(2-3):103–179, Jul 1992.

[43] P. Cousot and R. Cousot. A galois connection calculus for abstract interpretation. In *In Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '14. ACM, 2014.

[44] T. Cui and F. Franchetti. Autotuning a random walk boolean satisfiability solver. *Procedia Computer Science*, 4:2176–2185, 2011.

[45] David L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Automatic Verification Methods for Finite State Systems*, volume 407, pages 197–212. Springer-Verlag, 1989.

[46] J. Farkas. Uber die theorie der einfachen ungleichungen. *Journal für die Reine und Angewandte Mathematik*, 1902(124):1–27, 1902.

[47] Tomas Feder and Moshe Y. Vardi. The computational structure of monotone monadic snp and constraint satisfaction: A study through datalog and group theory. *SIAM Journal on Computing*, 28:57–104, 1998.

[48] Jean Claude Fernandez, Hubert Garavel, Alain Kerbrat, Laurent Mounier, Radu Mateescu, and Mihaela Sighireanu. Cadp a protocol validation and verification toolbox. In Rajeev Alur and Thomas A. Henzinger, editors, *Computer Aided Verification*, pages 437–440, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[49] Jean Claude Fernandez, Claude Jard, Thierry Jéron, and César Viho. Using on-the-fly verification techniques for the generation of test suites. In Rajeev Alur and Thomas A. Henzinger, editors, *Computer Aided Verification*, pages 348–359, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[50] D. M. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. In *Proc. of Conference Record of the 7th ACM Symposium on Principles of Programming Languages*, POPL'80, page 163–173. ACM Press, January 1980.

[51] GAO. Patriot missile defense : Software problem led to system failure at dhahran, saudi arabia. Technical report, U.S. General Accouting Office, 1992.

[52] P. Gastin, M. Mukund, and K. N. Kumar. Reachability and boundedness in time-constrained msc graphs. In *Perspectives in Concurrency Theory*, page 157–183. Universities Press, 2009.

[53] E. Goubault and S. Putot. Static analysis of numerical algorithms. In Kwangkeun Yi, editor, *Proceedings of the 13th International Static Analysis Symposium*, volume 4134 of *SAS'06*, pages 18–34, Berlin, Heidelberg, 2006. Springer, Heidelberg.

[54] N. Halbwachs, Y.-E. Proy, and P. Roumanoff. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design*, 11(2):157–185, Aug 1997.

[55] H. M. Hanisch. Analysis of place/transition nets with timed-arcs and its application to batch process control. In *Application and Theory of Petri Nets*, volume 691, page 282–299. LNCS, 1993.

[56] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model-checking for real-time systems. *Information and Computation*, 111:193–244, 1994.

[57] Thomas A. Henzinger, P. Ho, and H. Wong-Toi. Hytech: A model checker for hybrid systems. In Orna Grumberg, editor, *Computer Aided Verification*, pages 460–463, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[58] Anders Hessel and Paul Pettersson. A global algorithm for model-based test suite generation. *Electronic Notes in Theoretical Computer Science*, 190(2):47 – 59, 2007. Proceedings of the Third Workshop on Model Based Testing (MBT 2007).

[59] T. Hune, J. Romijn, M. Stoelinga, and F. Vaandrager. Linear parametric model checking of timed automata. *In Tools and Algorithms for the Construction and Analysis of Systems, Lecture Notes in Computer Science*, 2031, 2001.

[60] T. Hune, J. Romijn, M. Stoelinga, and F. Vaandrager. Linear parametric model checking of timed automata. *In Tools and Algorithms for the Construction and Analysis of Systems, Lecture Notes in Computer Science*, 2031, 2001.

[61] O. H. Ibarra, T. Jiang, N. Q. Trân, and H. Wang. New decidability results concerning two-way counter machines and applications. In *International Colloquium on Automata, Languages, and Programming (ICALP'93)*, volume 700 of *Lecture Notes in Computer Science*. Springer, 1993.

[62] Claude Jard and Thierry Jéron. Tgv: theory, principles and algorithms. *International Journal on Software Tools for Technology Transfer*, 7(4):297–315, Aug 2005.

[63] B. Jeannet and A. Miné. Apron: A library of numerical abstract domains for static analysis. In *Proceedings of International Conference on Computer-Aided Verification*, volume 5643, pages 661–667. Springer, Heidelberg, 2009.

[64] N. D. Jones, L. H. Landweber, and Y. E. Lien. Complexity of some problems in petri nets. *Theoretical Computer Sciences*, 4, 1977.

[65] A. Jovanovic, D. Lime, and O. H. Roux. Integer parameter synthesis for timed automata. *In TACAS, Lecture Notes in Computer Science*, 7795, 2013.

[66] M. Knapik and W. Penczek. *Bounded model checking for parametric timed automata*, pages 141–159. Springer, Heidelberg, 2012.

[67] Andrei Krokhin and Stanislav Zivn. *The Constraint Satisfaction Problem: Complexity and Approximability*, volume 7 of *Dagstuhl Follow-Ups*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2017.

[68] J. Larrosa and T. Schiex. Solving weighted csp by maintaining arc consistency. *Artificial Intelligence*, 159(1-2):1–26, Nov 2004.

[69] C. Lecoutre, F. Boussemart, and F. Hemery. Exploiting multidirectionality in coarse-grained arc consistency algorithms. In *Proceedings of Principles and Practice of Constraint Programming*, volume 4134 of *CP'03*, pages 480–494, Kinsale, Ireland, 2003. Springer, Heidelberg.

[70] N. G. Leveson and C. S. Turner. Investigation of the therac-25 accidents. *IEEE Computer*, 26(7):18–41, 1993.

[71] F. Logozzo and M. Fähndrich. Pentagons: A weakly relational abstract domain for the efficient validation of array accesses. *Science of Computer Programming Journal*, 75(9):796–807, Sep 2010.

[72] F. Lorber, A. Rosenmann, D. Ničković, and B. Aichernig. Bounded determinization of timed automata with silent transitions. *Real-Time Systems*, 53(3):291–326, May 2017.

[73] Kwiatkowska M., Norman G., and Parker D. Symbolic verification and strategy synthesis for linearly-priced probabilistic timed automata. In Bacci G. Ingólfsdóttir A. Legay A. Mardare R. Aceto L., Bacci G., editor, *Models, Algorithms, Logics and Tools*, volume 10460 of *Lecture Notes in Computer Science*, pages 289–309. Springer, Cham, 2017.

[74] Akan K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, Feb 1977.

[75] P. M. MERLIN. *A Study of the Recoverability of Computing Systems*. PhD thesis, Univ. California, Irvine, 1974.

[76] A. Miné. The octagon abstract domain. In *Proc. of Analysis, Slicing and Tranformation, in conjunction with Working Conference on Reverse Engineering*, AST'01, pages 310–319. IEEE CS Press, 2001.

[77] A. Miné. A new numerical abstract domain based on difference bound matrices. In *Proceedings of Programs as Data Objects II*, volume 2053 of *PDO'01*, pages 155–172. Springer, 2001.

[78] A. Miné. The octagon abstract domain. In *Proceedings of the 8th Working Conference on Reverse Engineering*, WCRE'01, pages 310–319. IEEE Comput. Soc, 2001.

[79] Th. S. Motzkin. *Beiträge zur Theorie der linearen Ungleichungen.* Doctoral thesis, University of Basel, Petersplatz 1, 4001 Basel, Suisse, 1933. English translation: Contributions to the theory of linear inequalities, RAND Corporation Translation 22, by D. R. Fulkerson, 1952.

[80] J. Ouaknine and J. Worrell. On the language inclusion problem for timed automata: closing a decidability gap. In *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science, 2004.*, pages 54–63, July 2004.

[81] R. Oucheikh, I. Berrada, and O. El Hichami. The 4-octahedron abstract domain. In Parosh Aziz Abdulla and Carole Delporte-Gallet, editors, *Networked Systems*, pages 311–317, Cham, 2016. Springer International Publishing.

[82] B. Parquier, L. Rioux, R. Henia, R. Soulat, Olivier H. Roux, and Étienne Lime, D.and André. Applying parametric model-checking techniques for reusing real-time critical systems. In Cyrille Artho and Peter Csaba Ölveczky, editors, *Formal Techniques for Safety-Critical Systems*, pages 129–144, Cham, 2017. Springer International Publishing.

[83] Ron Patton. *Software Testing.* Indianapolis: Sams Publishing, 2005. 978-0672327988.

[84] M. Péron and N. Halbwachs. An abstract domain extending difference-bound matrices with disequality constraints. In Byron Cook and Andreas Podelski, editors, *Proceed-*

ings of the 8th International Conference on Verification, Model Checking, and Abstract Interpretation, VMCAI'07, pages 268–282. Springer, Heidelberg, 2007.

[85] S. Porschen, E. Speckenmeyer, and X. Zhao. Linear cnf formulas and satisfiability. *Discrete Applied Mathematics*, 157(5):1046–1068, Mar 2009.

[86] Michael O. RABIN and Dana SCOTT. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125, 1959.

[87] C. Ramchandani. Analysis of asynchronous concurrent systems by timed petri nets, 1974.

[88] Michel A. Reniers and M. Van Weerdenburg. Action abstraction in timed process algebra. In Farhad Arbab and Marjan Sirjani, editors, *International Symposium on Fundamentals of Software Engineering*, pages 287–301, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[89] chairman: J-L Lions Report by the Inquiry Board. Ariane 5 – flight 501 failure. 1996.

[90] N. Roohi, P. Prabhakar, and M. Viswanathan. Robust model checking of timed automata under clock drifts. In *Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control*, HSCC '17, pages 153–162, New York, NY, USA, 2017. ACM.

[91] Tripakis S. Folk theorems on the determinization and minimization of timed automata. In Niebert P. Larsen K.G., editor, *Formal Modeling and Analysis of Timed Systems, FORMATS'03*, volume 2791 of *Lecture Notes in Computer Science*, Berlin, Heidelberg, 2004. Springer.

[92] S. Sankaranarayanan, H. Sipma, , and Z. Manna. Scalable analysis of linear systems using mathematical programming. In Radhia Cousot, editor, *Proceedings of the 6th International Conference on Verification, Model Checking and Abstract Interpretation*, volume 3385 of *VMCAI'05*, pages 21–41. Springer, Jan 2005.

[93] C. Sathawornwichit, T. Aoki, and T. Katayama. Modeling of real-time system designs for parametric analysis. In *2010 IEEE 16th International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 81–91, Aug 2010.

[94] Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of 10th ACM Symposium on Theory of Computing*, STOC'78, pages 216–226, New York, NY, USA, 1978. ACM Press.

[95] Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, STOC '78, pages 216–226, New York, NY, USA, 1978. ACM.

[96] R. F. Lutje Spelberg, R. C. M. de Rooij, and W. J. Toetenel. Experiments with parametric verification of real-time systems. In *Real-Time Systems, 1999. Proceedings of the 11th Euromicro Conference on*, pages 123–130, 1999.

[97] M. H. Sqalli, L. Purvis, and E. C. Freuder. Survey of applications integrating constraint satisfaction and case-based reasoning. In *Proceedings of the First International Conference and Exhibition on the Practical Application of Constraint Technologies and Logic Programming*, PACLP'99, 1999.

[98] S. Tripakis and S. Yovine. Analysis of timed systems using time-abstracting bisimulations. *Formal Methods in System Design*, 18(25–68), 2001.

[99] Moshe Y. Vardi. Branching vs. linear time : Final showdown. In Tiziana Margaria and Wang Yi, editors, *Proc. 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume Lecture Notes in Computer Science, 2031 of *TACAS'01*, pages 1–22. Springer-Verlag, April 2001.

[100] S. Verwer, M. de Weerdt, , and C. Witteveen. The efficiency of identifying timed automata and the power of clocks. *Information and Computation*, 209(3):606–625.

[101] Y. Wang. Annals of software engineering. *14*, 1–4:235–274, December 2002.

[102] Y. Wen, G. Li, and S. Yuen. On reachability analysis of updatable timed automata with one updatable clock. In *International Workshop on Structured Object-Oriented Formal Language and Method*, volume 9559 of *Lecture Notes in Computer Science*, pages 147–161. Springer, 01 2016.

[103] S. Yovine. Model-checking timed automata. *In School on Embedded Systems, Lecture Notes in Computer Science*, 1494, 1998.

[104] S. Kronos Yovine. A verification tool for real-time systems. *Journal of Software Tools for Technology Transfer*, 1(1-2):123–133, October 1997.

[105] C. Zhang, Q. Lin, L. Gao, and X. Li. Backtracking search algorithm with three constraint handling methods for constrained optimization problems. *Expert Systems with Applications*, 42(21):7831–7845, Nov 2015.

[106] D. Zhou, J. Huang, and B. Schölkopf. Learning with hypergraphs: Clustering, classification, and embedding. In P. B. Schölkopf, J. C. Platt, and T. Hoffman, editors, *Proceedings of Advances in Neural Information Processing Systems 19*, pages 1601–1608. MIT Press, 2006.

[107] W. M. Zuberek. Timed petri nets in modeling and analysis of manufacturing systems. *Journal of Factory Automation, Robotics and Soft Computing*, 1(2):30–37, 2007.

List of Figures

List of Tables

Résumé :

Les erreurs dues aux systèmes complexes peuvent causer d'énormes dégâts, des crises financières et même des pertes humaines. Par conséquent, leur vérification est la tâche principale dans leur processus de conception. Nous nous intéressons aux systèmes temps-réel paramétriques qui ont les contraintes de temps comme caractéristique principale. Dans cette thèse, nous construisons de nouvelles techniques de vérification basées sur des Automates Temporisés Paramétrés (TPA) comme outil de modélisation, et le Problème de Satisfaction de Contraintes (CSP) formalisme comme cadre de calcul pour la tractabilité des contraintes temporelles.

Dans cette thèse, nous traitons une sous-classe de CSP, appelée *4-CSP*. Nous fournissons les premières preuves de la traitabilité de *4-CSP*, et nous élaborons un algorithme de résolution de 4-CSP basé sur les théories de fermeture et de dépendance linéaire positive combinées à la technique de propagation des contraintes. La complexité temporelle et spatiale des algorithmes de résolution est démontrée être polynomiale et la comparaison avec d'autres travaux est bien discutée. Enfin, nous montrons l'importance de ces techniques dans de nombreuses applications telles que la vérification des systèmes temps-réel paramétriques et l'analyse statique de code. Principalement, nous utilisons cette puissance de calcul pour étudier la tractabilité des contraintes paramétrées de temps pour les systèmes modélisés par PTA, fournissant ainsi les algorithmes de l'analyse d'accessibilité en avant et en arrière.

Abstract:

The system verification is a primary task in their design process. We are interested in parametric real-time systems that have timing constraints as their main feature. In this thesis, we build new techniques for verification based on Parametric Timed Automata (PTA) as modeling tool, and Constraint Satisfaction Problems (CSPs) formalism as computation method for timing constraints resolution.

In this thesis, we deal with a CSP subclass, called 4-CSP. We provide the first graph-based proofs of the 4-CSP tractability, and we elaborate a 4-CSP resolution algorithm based on the closure and the positive linear dependence theories coupled with the constraint propagation technique. The time and space complexities of resolution algorithms are shown to be polynomial and the comparison with other works is discussed. Finally, we show the importance of these techniques in many applications such as parametric real-time verification and static code analysis. Mainly, we use this computation power to track the tractability of parametric timing constraints of systems modeled by PTA, providing thereby the forward and backward reachability analysis.