Université Paris Cité

INSTITUT DE RECHERCHE EN INFORMATIQUE FONDAMENTALE

UNIVERSITÉ PARIS CITÉ

Laboratoire IRIF

ALGORITHMIC TECHNIQUES FOR THE VERIFICATION OF COUNTER SYSTEMS AND PARAMETERISED NETWORKS

ARNAUD SANGNIER

Habilitation à Diriger des Recherches

Jury :

Rapporteurs :

Christel Baier	-	Professeure, Technische Universität Dresden, Allemagne
Javier Esparza	-	Professeur, Technische Universität Munchen, Allemagne
Jérôme Leroux	-	Directeur de Recherche, CNRS, LABRI

Examinateurs :

Nathalie Bertrand	-	Directrice de Recherche, INRIA, IRISA
Véronique Bruyère	-	Professeure, Université de Mons, Belgique
Thomas Colcombet	-	Directeur de Recherche, CNRS, IRIF

ABSTRACT

Model-checking is a verification technique which is in the past has been successfully applied to verify automatically the behavior of finite state systems. This approach consists in modelling a computing system by a mathematical model, in translating its specification into a logical formalism and then in proposing algorithms to check whether a model satisfies a logical formula. When the considered models have an infinite number of states, this method can easily lead to undecidable model-checking problems and one has hence to find the right trade-off between the expressiveness of models and specification languages and the feasibility of the verification. In this thesis, I present my contributions to the field of verification of infinite states systems where I have considered two main families of models. The first one are counter systems which can be seen as programs manipulating variables (called counters) taking their value in the natural. The second one are parameterised networks which can be seen as an abstraction of distributed networks where the number of participating entities is not fixed a priori and is unbounded. For these different models, I study exhaustively when the automatic verification is feasible and in the positive cases I try to design model-checking algorithms with optimal complexity bounds.

RÉSUMÉ

Le model-checking est une technique de vérification qui, par le passé, a été utilisée avec succès pour vérifier de façon automatique le comportement de systèmes à états finis. Cette approche consiste à modéliser un système informatique par un modèle mathématique, à traduire sa spécification dans un formalisme logique et ensuite à proposer des algorithmes pour s'assurer qu'un modèle satisfait une formule logique. Lorsque les modèles considérés ont un nombre infini d'états, cette méthode mène souvent à des problèmes de model-checking indécidables et pour cette raison, il faut trouver le bon compromis entre l'expressivité des modèles et des langages de spécification et la faisabilité de la vérification. Dans cette thèse, je présente mes contributions à la vérification de systèmes avec un nombre infini d'états dans lesquelles je me suis penché sur deux grandes familles de modèles. La première est celle des systèmes à compteurs qui peuvent être vus comme des programmes manipulant des variables à valeur entière (appelées compteurs). La deuxième famille est celle des réseaux paramétrés qui sont des abstractions de réseaux distribués dans lesquels le nombre de participants n'est pas fixé et est non borné. Pour ces différents modèles, j'étudie de façon exhaustive quand la vérification automatique est possible et dans les cas positifs j'essaie de concevoir des algorithmes de model-checking avec des bornes de complexité optimales.

FOREWORD

This document sums up some of the results I have obtained together with other researchers since I have defended my PhD in november 2008. My main goal when writing it was to provide an uniform formal framework to present my research, to give intuitions on the way I obtained some of the key results and to relate my works with the results close to what I have done. As a matter of fact, I did not write any formal proof in this document and I invite the reader to seek for the corresponding paper if he desires to read the technical developments of a result.

This thesis is divided into two main parts, each being composed of some chapters. At the beginning of each chapter, I recall in an informal way the main contributions it presents and to ease the reading, the main results I have obtained are all located in a (red) frame.

CONTENTS

1 INTRODUCTION

1.1 CONTEXT

1.1.1 *On the verification of computing systems*

Nowadays, computing systems are part of our everyday life and occur in most of the aspects of our society. We use them at a personal level, for instance through the different applications present in our smartphones or thanks to the different websites we consult on the Internet, but they play as well a crucial and central role in the industrial world, where they are used to control and monitor vehicles or complex machines, to perform measures allowing to understand better our world, to assist men in tasks needing an extreme precision. In the last thirty years, there have indeed been a blowup in the use of such systems, one of the goal being to improve our life and our society but as well to acquire more knowledge.

Since we rely a lot on them, it is hence extremely important to develop strong theoretical foundations in computer science to fully understand what can be or cannot be achieved with such systems and as well to improve them, as it is fundamental to ensure that they behave correctly, i.e. they are performing the tasks for what they have been designed for. Indeed, since these systems have been conceived by men, they can suffer from design or programming errors, commonly called as bugs. In some cases these bugs do not yield to big problems, for instance a bug in the graphical representation of messages in our phone, in other cases they can lead to terrible issues, as for instance a problem in a software responsible for automatically driving a car. In fact according to the application, the effort put in the verification of a system can be very different, as an example systems developed to monitor and control aircrafts while flying are designed following very strict rules which allow to ensure their correct behaviour with a very low rate of error (due most of the time to mechanical failures, one tries to avoid thanks to replication) but games or applications executing on our phones do not need to follow such a process and in fact we can often witness some errors in such applications.

The reason why all applications are not verified with the same meticulousness is that in order to offer some strong guarantees, this methodology has a cost. First, often restrictions need to be imposed in the design of the system to ensure the feasibility of the verification, for instance for many softwares used in critical systems, dynamic memory allocation is forbidden. Second, because of the complexity of the systems, the verification might take some times or even not terminate, due to the blowup in all the possible scenarios to explore. Indeed, to be certain that no bug may appear, it is necessary to take into account all the possible cases that will occur while the system will be used, in case of a system interacting with an environment this can become really tricky. Finally, one of the major issue to develop techniques for the automatic verification of systems is that one has to face with some undecidability results, which, to briefly sum up, tell us that it is impossible to design a program ensuring that any program behaves correctly.

1.1.2 *The model-checking approach*

These different problems are at the heart of my field of research: *the automatic verification of complex systems*, and more precisely I am focused on a specific technique of verification, which is known as *model-checking*. This technique consists in taking a mathematical model representing a system and a specification given in a logical formalism and to check mathematically, thanks to a model-checking algorithm, whether the model satisfies the specification. The development of this technique and its application to the verification of concrete systems correspond to a major progress in the field of theoretical computer science which has been acknowledged in 2007 by a Türing award given to Edmund M. Clarke, E. Allen Emerson and Joseph Sifakis for having make grow this approach. One of the major problem with the model-checking technique is to find the good trade-off between expressivity of the models and the specification languages and the feasibility and complexity of the model-checking. Assume for instance that the considered model are the Türing machines, then one knows that there does not exist any algorithm which takes as input any machine and tells whether it halts or not, this problem is indeed undecidable, however it is possible to check whether it halts after at most ten steps, but this latter property is less interesting. It is hence capital to understand what can or cannot be achieved with the model-checking approach, this means study different models and classes of specification languages and determine where lies the frontier between decidability and undecidability for the model-checking problem as well as establish its complexity in the case it is decidable.

Note that pursuing an exhaustive study on the model-checking problem is extremely important from a scientific point of view as it allows to improve the knowledge we have on what can or cannot be achieved with computer systems, but it leads as well to more practical developments. As a matter of fact, when the model-checking problem is undecidable, one technique consists in establishing decidable results for more abstract models that over approximate the behaviour of a system and which can guarantee the absence of bugs, but if a bug is detected, it does not mean that the system under study has one, it could be caused by the over approximation. Finally, the study of models allows as well to understand better the expressive power of some formalisms, and this information can be used at a design level to establish what are the better formal paradigms to rely on to build a system.

1.1.3 *From simple models to models with an infinite state space*

One of the simplest model considered in the model-checking approach are Kripke structures, they can be seen as finite state machines where each state (or configuration) is labelled by a set of atomic propositions characterising its properties. For this model, different specification languages have been developed, which allow to describe the evolution of the system during time. Some of them, called linear time specifications, are used to provide allowed sequences of states, aka as executions. They can be given either under the form of an automaton recognising such good sequences or a more user friendly logic based formalism as the Linear time Temporal Logic (LTL) . Other languages allow instead to speak about the possibility of the system to go either in a certain state or in another one, they are known as branching time specifications. They provide some sets of trees describing the allowed evolution of the system. Here again there exists some automaton based formalism as tree automata or some logical formalisms like the Computation Tree Logic (CTL). For both these families of formalisms, the model-checking problem, which asks whether the behaviour of a

Kripke structure belongs to the set of allowed behaviour given by an automaton or a logical formula, has been intensively studied in the end of the last century and its complexity is well known (see for instance [BK08] for a detailed study).

In order to model more complex systems, one can extend Kripke structures by equipping them with some storage mechanisms. Most of the time these storage systems allow to represent a specific aspect of a system being modelled and have an unbounded capacity. Note that even when these storage mechanisms are bounded, and hence the model under study remains with a finite set of configurations, syntactic use of such mechanisms can be relevant as the description of the model is often more succinct. For instance if a program performs a loop increasing at each turn an integer variable bounded by 2^{10}, the program can be described thanks to a few lines, whereas its behaviour is exponentially bigger. The storage mechanisms are added to Kripke structures by allowing the transitions of the structure to test and update the mechanisms while going from a control state to another one.

For example in [AD94], Alur and Dill have proposed to add clocks to finite state systems in order to model systems whose behaviour depends on the evolving of time. These clocks are variables taking their value into the positive reals and all evolving at the same rate. The transitions of the systems can reset some clocks, test their values and compare them one to another. This gives raise to the model of Timed Automata which have been then intensively studied both from the theoretical and practical point of views. The underlined semantics is given by a Kripke structure (or equivalently a transition system) with an infinite number of states, each state containing a control state of the timed automaton and a real value for each clock. Another class of systems which has been intensively studied in the field of verification are pushdown systems (see for instance [BEM97]). In that case, a stack is added to the finite state machine and the transitions can either push or pop symbols from a finite stack alphabet. This model is well suited to represent the behavior of programs with recursive calls, indeed the stack of the system can model the stack of the programs where the different return points of functions calls are stored. Pushdown systems have then been extended for instance by considering more than one stack to model concurrent programs with recursive calls as it is done in [Ati10]. Here again the underlying transition system associated to a pushdown system has an infinite number of configurations since there is no limit on the content of the stack. Another extension that has been considered consists in adding a FIFO queue instead of a stack as a storage mechanism, the transitions being able to put or to get elements from the queue as it is done in [BZ83]. This feature allows to model distributed systems communicating thanks to message passing. Another extensions that have been intensively studied are counter systems (see for instance [CJ98]), in that case the system manipulates variables taking their value in the naturals and it can test them or update their value. Here again such systems give rise to infinite behaviours if one does not put any bound on the counter values.

Furthermore other research works have considered ways to mix these storage mechanisms, or to assume that the data that can be stored belong not anymore to a finite domain but to an infinite one, it has been as well proposed to assume that the transitions relation of the systems is not non-deterministic but probabilistic or both of them. Indeed the field of possibilities to extend the simple model of Kripke structures is wide and the goals, when one studies the verification of systems, are both to understand which extensions make sense for practical uses and as well to understand where lies the frontier between expressivity of the models and feasibility (in the sense of decidability and complexity) of the verification. As

a matter of fact, a good candidate to model computing systems would be Türing machines, but it is well-known that in that case even a simple verification problem such as the halting problem is undecidable. Furthermore from this undecidability result, other undecidability results can be deduced as for instance for pushdown systems equipped with two stacks or for counter systems manipulating two counters that can be incremented, decremented and whose value can compared with 0 [Min67]. For other extensions, many verification problems are decidable as it is the case for timed automata, but it is as well important to understand the cost in terms of complexity of these problems and what are the features that make a problem complex to solve. For instance, in timed automata, if the considered automaton has a single clock then the reachability of a control state is NLOGSPACE-complete [LNS04] and this problem becomes PSPACE-complete as soon as the number of allowed clocks is bigger than two [AD94; FJ15].

My main research works have hence consisted in studying systems with an infinite state space and in establishing what are the characteristics of these models that render the automatic verification process feasible. I have focused my attention on two main families of models, the first one is the family of counter systems for which in their full generality model-checking problems are often undecidable and the second one is the family of parameterised networks where we consider networks of entities all executing the same protocol given by a finite state machine and where I looked at different means of communication.

1.1.4 *Infinite state systems studied in this thesis*

1.1.4.1 *Counter systems*

Counter systems are finite state automata equipped with program variables (counters) interpreted over non-negative integers. Each transition of the automaton can then test or update the different counter values and according to the allowed operations, different families of systems can be obtained. For example, one can authorise only simple decrements or increments of the counter values, or suppose that the updates can be performed with more complex functions or even that the change of the counter values follows a relation. The granularity of the considered operations depends on the system and on the specification one wishes to analyse; in some cases, a strong non-determinism is needed due to the uncertainty one has on the system, in other cases, one could desire to model the low-level operations which compose a complex update in a non atomic fashion. As we have already mentioned, in their full generality, these systems are Türing-complete models of computation, often used to describe the behaviour of complex real-life systems, such as embedded/control hardware and/or software systems. For instance, they can be used to represent the behaviour of programs working over integer variables (the other operation of the programs being abstracted). They can encode too executions of distributed networks thanks to a counting abstraction (which consists in forgetting the identities of the entities in the networks and only remember their number) [GS92]. In [FLS07; Bou+11], it has been shown that they allow to verify programs working over simple linked lists which manipulate dynamically the memory heap. Petri nets [Pet62] which are a model of practical use well adapted for the design of concurrent processes, can be encoded into Vector Addition System with States. Their verification is as well related with some other problems in formal methods as for instance the satisfiability of the first oder logic over data words with two variables which can be reduced to a reachability question over counter systems [Boj+11].

Because many verification problems of rather complex systems can be reduced to decision problems for counter systems, it is important to understand the difficulties faced by potential verification algorithms designed to work for this latter model. Due to their succinctness and expressive power, most decision problems, such as reachability, termination and temporal logic model-checking, are undecidable for counter systems, even when the operations on the counters are restricted to increment, decrement and zero-test [Min67]. This early negative result motivated the search for subclasses with decidable verification problems. Such classes include:

- One Counter Systems [Göl+10],
- Vector Addition Systems with States [Lip76] where the counters can only be incremented and decremented but not tested (these systems are equivalent to Petri Nets),
- Reversal-Bounded Counter Machines [Iba78], for this class a semantics restriction is imposed which states that during an execution for each counter the number of alternations between increasing and decreasing mode is bounded, and,
- Flat Counter Systems [CJ98; Boi99; FL02], for which each control state of the underlying finite state automaton belongs to at most one simple cycle.

During the last years, I have studied model-checking problems for many of these subclasses with decidable reachability problems considering different kind of specifications (going from linear time to branching time specifications). The most important results I have obtained are reported in the first part of this thesis.

1.1.4.2 *Parameterised networks*

Another family of infinite state systems I have been studying are what are called parameterised networks. In [GS92], German and Sistla introduced a model to represent networks with a fix but unbounded number of entities. In this model, each participant executes the same protocol and they communicate between each other thanks to rendez-vous (a synchronisation mechanism allowing two entities to change their local state simultaneously). The number of participants can then be seen as a parameter of the model and possible verification problems ask for instance whether a property holds for all the values of this parameter or seeks for some specific value ensuring a good behaviour. With the increasing presence of distributed mechanisms (mutual exclusion protocols, leader election algorithms, renaming algorithms, etc) in the core of our computing systems, there has been in the last two decades a regain of attention in the study of such parameterised networks.

Surprisingly, the verification of these parameterised systems is sometimes easier than the case where the number of participants is known. This can be explained by the following reason: in the parameterised case the procedure can adapt on demand the number of participants to build a problematic execution. It is indeed what happens with the liveness verification of asynchronous shared-memory systems. This problem is PSPACE-complete for a finite number of processes and in NP when this number is a parameter [Dur+17]. It is hence worth studying the complexity of the verification of such parameterised models and many recent works have attacked these problems considering networks with different means of communication. For instance in [EFM99; Ber+19; BBM21] the participants communicate thanks to broadcast of messages, in [Cla+04; Ami+14] they use a token-passing mechanism ,

in [BGS14] a message passing mechanism and in [EGM16] the communication is performed through shared registers. The relative expressiveness of some of those models has been studied in [ARZ15]. In the last years there are as well be a regain of attention on the analysis of population protocols, a model originally introduced by researchers in distributed computing [Ang+04] and which can be seen as a parameterised network with rendez-vous communication and fairness. The novelty with this latter model lies in the properties one wishes to ensure and this gives rise to new interesting problems to study on parameterised networks (see for instance [Esp+17; Blo+21; Esp+21]) both from the theoretical and the practical points of view. Finally in his survey [Esp14], Esparza shows that minor changes in the setting of parameterised networks, such as the presence of a controller (or equivalently a leader), might drastically change the complexity of the verification problems.

After my PhD, together with Giorgio Delzanno and Gianluigi Zavatarro, we introduced a model of parameterised networks inspired by Ad Hoc Networks. As a matter of fact, the two main features of this model are first that the communication should be performed by a broadcast primitive (to mimic the radio transmission in Ad Hoc Networks) and second that our model is equipped with a communication topology such that only the neighbours of an emitter can receive a broadcast message. This model extends hence the model of broadcast protocols introduced in [EFM99] by adding a communication topology. The second part of this thesis present the main results I have obtained in this context.

1.2 CONTRIBUTIONS

1.2.1 *Content of this thesis*

We sum up briefly here the contents of the different chapter of this thesis.

Chapter 2 provides a description of the main mathematical notations and tools we use in this document.

In Chapter 3 we present formally the general model of counter systems we consider in this thesis and that we call Affine Counter Systems where the tests allowed on the counter values are quantifier free formulas of the Presburger arithmetic and the updates are given by affine functions. We introduce then different restrictions of this model and recall some general results about the reachability problems of a control state and of a configuration for the introduced restrictions.

In Chapter 4, we present the results we have obtained for the model-checking of flat Affine Counter Systems with linear time specifications, i.e. specifications describing sets of allowed executions. One interesting point concerning our specification languages is that they can use as atomic propositions constraints about the values of the counters. It was known before, see [Dem+10], that the model-checking problem for flat Affine Counter Systems with the finite monoid property is decidable for a very large class of specifications but the precise complexity was not known and the technique which consists in translating the model-checking problem into the satisfiability problem for a formula of Presburger arithmetic was far from optimal. We present here tight complexity results for the model-checking problems of a wide range of linear time specification languages going from the linear time temporal logic with past to the linear μ-calculus. More interestingly, we provide a general algorithmic method that can be instantiated for each of the considered specification languages and that could be reused in some other contexts. Many results described in this chapter were

obtained during the PhD of Amit Kumar Dhar that I co-supervised together with Stéphane Demri. These results have been published in the following papers: [DDS12; DDS15; DDS13; IS16]

In Chapter 5, we present results for the model-checking of counter systems taking as specification language the linear time temporal logic with the freeze quantifier (Freeze LTL) [DL09]. This logic, originally developed to describe infinite data words (i.e. words where at each position there is a letter from a finite alphabet and a datum from an infinite alphabet) allows, when used to describe runs of counter systems, to compare the values of the counters between different configurations. Indeed this logic is equipped with an operator allowing to store at some point of an execution the value of a counter in a register and to compare later on whether the stored value is equal or not with the value of another counter. One can for instance specify properties as: there exists an execution such that after some point, the first counter will always take different values. Such properties cannot be express with the specification languages presented in Chapter 4. As a matter of fact, the model-checking problem for Freeze LTL is much more complex to analyse. I have begun to study the model-checking problem for Freeze LTL over counter systems during my PhD thesis together with Ranko Lazic and Stéphane Demri [DLS10] focusing on One Counter Systems and we showed that in general this problem is undecidable but that decidability can be regained by considering restrictions on the systems (for instance assuming the counter system is deterministic) or on the logic (with the flat fragment of Freeze LTL). We consider in this chapter other classes of counter systems and study the decidability status of the model-checking problem taking into account various hypothesis on the model. We furthermore provide a tight complexity bound for what concerns the verification of one counter systems with the flat fragment of Freeze LTL. The presented results come from the following papers: [DS10; BQS17; BQS19].

In Chapter 6, we present the results we have obtained concerning the model-checking of branching time temporal logics over counter systems. We first show that for flat counter systems whose update are translations, the model-checking problem of the temporal logic CTL* is interreducible to the satisfiability problem for Presburger arithmetic formulas and hence harder than the model-checking of all the linear time specifications we have studied in Chapter 4. In the second part, we study the model-checking of the modal μ-calculus over Vector Addition Systems with States (VASS). It is well known that over this class of model, most model-checking problems of linear time specifications are decidable and when one considers branching time temporal logics, the model-checking problem becomes quickly undecidable (even if the logics is not very expressive) (see e.g. [EN94]). This latter result has discouraged research works in this direction. However, we succeed in defining a fragment of the modal μ-calculus for which the model-checking problem over VASS is decidable. Our proof technique uses a reduction to a specific kind of turn based two-player games played on the transition system of the VASS for which we show that we can compute the winning region for the first player. Finally, we prove that our logics can be used to solve some qualitative problems over VASS extended with probabilities and non-determinism, hence its expressive power is of practical use. The results obtained in this chapter are extracted from the following papers: [DDS14; DDS18; Abd+13; Abd+16a].

In Chapter 7, we introduce our model of parameterised networks with broadcast and communication topology that is inspired from ad hoc networks. We call in fact Ad Hoc Networks these networks. In this chapter, we assume that the communication topology does

not change (which means that there is no mobility in the modelled network). In our model, we suppose that each entity in the network executes the same finite state protocol, a finite state machine whose transitions are labelled by broadcast or reception of messages from a finite message alphabet. Among the different verification problems we consider, a central one is the parameterised reachability of a control state, it asks whether there exists an initial topology (where all the entities are in the initial state of the protocol), from which one can find an execution ending in a configuration where one of the entity ends in a specific state. When the initial topology is given, this problem boils down to the analysis of a finite state system, the difficulty comes here from the fact that we need to seek for the initial topology. We show that this problem is undecidable but that decidability can be regained by imposing some restrictions on the allowed communication topologies. In the last part of the chapter, we propose an extension of our model with timed constraints à la timed automata and we see in which measure we can obtain decidability of the parameterised reachability problem in this context. These results have been presented in the following works: [DSZ10; DSZ11; Abd+11; Abd+16b].

In Chapter 8, we study again the model of Ad Hoc Networks of the previous chapter but with different hypothesis concerning the communication topology. We assume here that it can change in a complete non deterministic manner at any instant, the number of entities remaining constant. This feature models in a very abstract way the mobility in a network. This gives rise to the model we call Reconfigurable Broadcast Network. Whereas in Ad Hoc Networks, simple reachability questions are undecidable, it is not anymore the case in Reconfigurable Broadcast Networks and we even show that the parameterised reachability of a control state can be solved in polynomial time. Encouraged by this positive result, I have studied other problems on Reconfigurable Broadcast Networks. For instance, together with Nathalie Bertrand and her PhD student Paulin Fournier that I partially co-supervised, we studied a local semantics for such networks where we ask that each entity takes the same decision according to its past actions (for instance in the initial state, it is not possible that one entity broadcasts a message $m1$ and another one a message $m2$ if they both have not done anything before). We proved that under this restriction the parameterised control state reachability is still decidable. We studied as well a probabilistic version of the Reconfigurable Broadcast Networks, where the protocol executed by each entity is equipped with some probabilistic choices and we proposed algorithmic solutions to solve qualitative parameterised problems on this model. Finally, in the last part of the chapter, I present an extension of Reconfigurable Broadcast Networks where the transmitted messages can belong to an infinite alphabet to model for instance the fact that entities in the network can send their identities to other ones. In this context, we provide a characterisation of the decidability status of the parameterised control state reachability problem and show that the number of data entities allowed to be exchanged in a same message is a crucial factor. The results presented in this chapter are extracted from the following works: [DSZ10; Del+12; BFS15; BFS14; DST13; DST16].

1.2.2 *Other contributions*

In this document, I have presented my research following the two main directions I have been working on since I defended my PhD. However I have done some other works not included in this thesis that I briefly mention here.

Verification of Time Petri Nets. In [RS09], together with Pierre-Alain Reynier, we studied Time Petri Nets, which are Petri Nets extended with temporal constraints. In this model, a clock is associated to each transition which is reset when the transition becomes newly enabled (i.e. there are enough tokens to fire the transitions) and furthermore the transitions are labelled with timed intervals specifying when the transitions should be fire. In the classical semantics, even simple reachability questions such as the coverability problem are undecidable, mostly due to the fact that when a transition can be fired, it has to be either disabled by the firing of another transition or fired (this is called the urgent semantics). We show that when this urgency policy is relaxed, it is possible to regain decidability.

Verification of Graph Transformation Systems. In [Ber+12], we studied decidability issues for reachability problems in graph transformation systems, a powerful infinite-state model described by rules matching some patterns in a graph and changing it by adding or deleting some edges and vertices. For a fixed initial graph, we looked at different types of reachability questions: reachability of an entirely specified graph or of a graph that satisfies a given pattern (coverability). We reformulated results obtained, e.g., for context-free graph grammars and concurrency models, such as Petri nets, in the more general setting of graph transformation systems and studied new results for classes of models obtained by adding constraints on the form of reduction rules.

Verification of parameterised networks with shared memory. In [Bou+16], we studied a family of parameterised networks where the communication takes place thanks to a shared register in which each entity can read or write a datum from a finite given set. This model has been studied before in [EGM16]. We looked at the almost-sure reachability problem for this system where we assumed that the non-determinism is resolved by a stochastic scheduler. Given a protocol, we focused on almost-sure reachability of a target state by one of the processes. The answer to this problem naturally depends on the number N of processes. However, we proved that our setting has a cut-off property: the answer to the almost-sure reachability problem is constant when N is large enough; we then developed an EXPSPACE algorithm deciding whether this constant answer is positive or negative.

Model-checking of counting temporal logics. Together with Peter Habermehl, I visited twice the Institute for Software Engineering and Programming Languages of the University of Lübeck to work on extensions of (linear time and branching time) temporal logics where it is possible to count how often a certain properties holds. Our results are sum up in [Dec+17] where we show that for most of the extensions the model-checking problem is undecidable, but that decidability can be recovered by considering flat Kripke structures where each state belongs to at most one simple loop. Most decision procedures are based on results on (flat) counter systems.

Verification of algorithms for robots evolving on a ring. In [San+17; San+20], together with Nathalie Sznajder and researchers in distributed computing, we studied verification problems for autonomous swarms of mobile robots that self-organize and cooperate to solve global objectives. In particular, we focused on a model where anonymous robots evolve on a finite ring. A large number of algorithms have been proposed working for rings whose size is not a priori fixed and can be hence considered as a parameter. Handmade correctness proofs of these algorithms have been shown to be error-prone. We hence looked at the feasibility of the automatic verification problem of such algorithms in the parameterised case. We showed that safety and reachability problems are undecidable for robots evolving asynchronously. On the positive side, we showed that safety properties are decidable in the synchronous case,

as well as in the asynchronous case for a particular class of algorithms. Decision procedures rely on an encoding into the satisfiability problem Fr Presburger arithmetics formulas that can be verified by an SMT-solver. We proposed as well an encoding of several case studies in order to check whether our approach is feasible in practice.

Verification of parameterised networks with rendez-vous. Together with Florian Horn, we studied some verification problems in parameterised networks where the communication is performed thanks to pairwise rendez-vous (an entity emits a message that is received by another one, and an emitter can send a message only if there is a receiver). This was the model originally considered in the seminal work on parameterised verification by German and Sistla [GS92]. In [HS20], we looked at a new problem on this model which consists in asking whether there exists a bound B such that in all networks with more than B entities, there exists an execution where all entities end in a final state. We called such a bound a cut-off. We provided decidability and complexity results for this problem under various assumptions, such as absence/presence of a leader or symmetric/asymmetric rendez-vous.

Formal methods for the design of Distributed Algorithms. Together with Benedkit Bollig, in the context of the project FREDDA (FoRmal mEthods for the Design of Distributed Algorithms) financed by the french national research agency ANR, we studied different models and specification languages for analysis of distributed algorithms. In [BRS21], we introduced Distributed Memory Automata, a model of register automata suitable to capture some features of distributed algorithms designed for shared memory systems. In this model, each participant owns a local register and a shared register and has the ability to change its local value, to write it in the global memory and to test atomically the number of occurrences of its value in the shared memory, up to some threshold. One important fact is that the set of values is possibly infinite. We obtained some decidability results for the control state reachability problem in such context. In [BSS21], we decided to investigate specification languages for distributed algorithms manipulating data (as for instance the identities of the processes). With this in mind, we studied first-order logic over unordered structures whose elements carry two data values from an infinite domain which can be compared with equality. The idea behind this specification language is that each element represents a participant of a distributed algorithm, its first data being its input value and its second data its output value. This formalism is hence suitable to specify the input-output behaviour of various distributed algorithms. As the logic is undecidable in general, we introduced a family of local fragments that restrict quantification to neighbourhoods of a given reference point and studied where lies the frontier between decidability and undecidability of the satisfiability problem for these fragments.

1.2.3 *List of publications*

International Journals

[San+20] Arnaud Sangnier, Nathalie Sznajder, Maria Potop-Butucaru and Sébastien Tixeuil. *Parameterized verification of algorithms for oblivious robots on a ring. Formal Methods in System Design* 56(1), pages 55-89. Springer, 2020.

[BQS19] Benedikt Bollig, Karin Quaas and Arnaud Sangnier. *The Complexity of Flat Freeze LTL. Logical Methods in Computer Science* 15(3), 2019.

[DDS18] Stéphane Demri, Amit Kumar Dhar and Arnaud Sangnier. *Equivalence between model-checking flat counter systems and Presburger arithmetic. Theoretical Computer Science* 735, pages 2-23, 2018.

[DST16] Giorgio Delzanno, Arnaud Sangnier and Riccardo Traverso. *Adding Data Registers to Parameterized Networks with Broadcast. Fundamenta Informaticae* 143(3-4), pages 287-316. 2016.

[Abd+16b] Parosh Aziz Abdulla, Giorgio Delzanno, Othmane Rezine, Arnaud Sangnier and Riccardo Traverso. *Parameterized verification of time-sensitive models of ad hoc network protocols. Theoretical Computer Science* 612 (1-22). Elsevier Science Publishers, 2016.

[DDS15] Stéphane Demri, Amit Kumar Dhar and Arnaud Sangnier. *Taming past LTL and flat counter systems. Information & Computation* 242, pages 306-339, 2015.

[DLS10] Stéphane Demri, Ranko Lazić and Arnaud Sangnier. *Model checking memoryful linear-time logics over one-counter automata. Theoretical Computer Science* 411 (22-24), pages 2298-2316. Elsevier Science Publishers, 2010.

International Conferences

[BSS21] Benedikt Bollig, Arnaud Sangnier, and Olivier Stietel. *Local First-Order Logic with Two Data Values.*. In *Proceedings of the 41st International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'21)*, LIPIcs 213, pages 39:1-39:15. Leibniz-Zentrum fur Informatik, 2021.

[BRS21] Benedikt Bollig, Fedor Ryabinin and Arnaud Sangnier. *Reachability in Distributed Memory Automata*. In *Proceedings of the 29th EACSL Annual Conference on Computer Science Logic (CSL'21)* , LIPIcs 183, pages 13:1-13:16. Leibniz-Zentrum fur Informatik, 2021.

[HS20] Florian Horn and Arnaud Sangnier. *Deciding the existence of cut-off in parameterized rendez-vous networks.* In *Proceedings of the 31st International Conference on Concurrency Theory (CONCUR'20)* , LIPIcs 171, pages 46:1-46:16. Leibniz-Zentrum fur Informatik, 2020.

[Del+19] Carole Delporte-Gallet, Hugues Fauconnier, Yan Jurski, François Laroussinie and Arnaud Sangnier. *Towards synthesis of distribued algorithms with SMT solvers* In *Proceedings of the 7th International Conference on NETworked sYStems (NETYS'19)*. LNCS 11704, pages 200-216. Springer, 2019.

[San+17] Arnaud Sangnier, Nathalie Sznajder, Maria Potop-Butucaru and Sébastien Tixeuil. *Parameterized verification of algorithms for oblivious robots on a ring.* In *Proceeding of 2017 Formal Methods in Computer Aided Design (FMCAD'17)*. IEEE, 2017.

[BQS17] Benedikt Bollig,Karin Quaas and Arnaud Sangnier. *The Complexity of Flat Freeze LTL.* In *Proceedings of the 28th International Conference on Concurrency Theory (CONCUR'17)* , LIPIcs 85, pages 33:1-33:16. Leibniz-Zentrum fur Informatik, 2017.

[Dec+17] Normann Decker, Peter Habermehl, Martin Leucker, Arnaud Sangnier and Daniel Thomas. *Model-checking Counting Temporal Logics on Flat Structures* . In *Proceedings of the 28th International Conference on Concurrency Theory (CONCUR'17)* , LIPIcs 85, pages 29:1-29:17. Leibniz-Zentrum fur Informatik, 2017.

[IS16] Radu Iosif and Arnaud Sangnier. *How Hard is It to Verify Flat Affine Counter Systems with the Finite Monoid Property ?.* in *Proceedings of the 14th International Symposium on Automated Technology for Verification and Analysis (ATVA'16)* , LNCS 9938, pages 89-105. Springer, 2016.

[Bou+16] Patricia Bouyer, Nicolas Markey, Mickael Randour, Arnaud Sangnier and Daniel Stan. *Reachability in Networks of Register Protocols under Stochastic Schedulers.* In *Proceedings of the 43th International Colloquium on Automata, Languages and Programming - Part II (ICALP'16 (2))*, LIPIcs 55, pages 106:1-106:14. Leibniz-Zentrum fur Informatik, 2016.

[Abd+16a] Parosh Aziz Abdulla, Radu Ciobanu, Richard Mayr, Arnaud Sangnier and Jeremy Sproston. *Qualitative Analysis of VASS-Induced MDPs.* In *Proceedings of the 19th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'16)*, LNCS 9634, pages 319-334. Springer, 2016.

[BFS15] Nathalie Bertrand, Paulin Fournier and Arnaud Sangnier. *Distributed Local Strategies in Broadcast Networks.* In *Proceedings of the 26th International Conference on Concurrency Theory (CONCUR'15)*, LIPIcs 42, pages 44-57. Leibniz-Zentrum fur Informatik, 2015.

[LMS15] François Laroussinie, Nicolas Markey and Arnaud Sangnier. *ATLsc with partial observation.* In *Proceedings of the 6th International Symposium on Games, Automata, Logics and Formal Verification (GandALF'15)*, EPTCS 193, pages 43-57. 2015.

[DDS14] Stéphane Demri, Amit Kumar Dhar and Arnaud Sangnier. *Equivalence Between Model-Checking Flat Counter Systems and Presburger Arithmetic.* In *Proceedings of the 8th International Workshop on Reachability Problems (RP'14)*, LNCS 8762, pages 85-97. Springer, 2014.

[BFS14] Nathalie Bertrand, Paulin Fournier and Arnaud Sangnier. *Playing with probabilities in Reconfigurable Broadcast Networks.* In *Proceedings of the 17th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'14)*, LNCS 8412, pages 134-148. Springer, 2014.

[DST13] Giorgio Delzanno, Arnaud Sangnier and Riccardo Traverso. *Parameterized Verification of Broadcast Networks of Register Automata.* In *roceedings of the 7th International Workshop on Reachability Problems (RP'13)*, LNCS 8169, pages 109-121. Springer, 2013.

[Abd+13] Parosh Abdulla, Richard Mayr, Arnaud Sangnier and Jeremy Sproston. *Solving Parity Games on Integer Vectors.* In *Proceedings of the 24th International Conference on Concurrency Theory (CONCUR'13)*, LNCS 8052, pages 106-120. Springer, 2013.

[DDS13] Stéphane Demri, Amit Kumar Dhar and Arnaud Sangnier. *On the Complexity of Verifying Regular Properties on Flat Counter Systems.* In *Proceedings of the 40th International Colloquium on Automata, Languages and Programming - Part II (ICALP'13 (2))*, LNCS 7966, pages 162-173. Springer, 2013.

[Del+12] Giorgio Delzanno, Arnaud Sangnier, Riccardo Traverso and Gianluigi Zavattaro. *On the Complexity of Parameterized Reachability in Reconfigurable Broadcast Networks.* In *Proceedings of the 32nd International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'12)*, LIPIcs 18, pages 289-300. Leibniz-Zentrum fur Informatik, 2012.

[DDS12] Stéphane Demri, Amit Kumar Dhar and Arnaud Sangnier. *Taming Past LTL and Flat Counter Systems.* In *Proceedings of the 6th International Joint Conference on Automated Reasoning (IJCAR'12)*, LNAI 7634, pages 179-193. Springer, 2012.

[DSZ12] Giorgio Delzanno, Arnaud Sangnier and Gianlugi Zavattaro. *Verification of Ad Hoc Networks with Node and Communication Failures.* In *Proceedings of the 32nd International Conference on Formal Techniques for Distributed Systems (FMOODS/FORTE'12)*, LNCS 7273, pages 235-250. Springer, 2012.

[Ber+12] Nathalie Bertrand, Barbara König, Giorgio Delzanno, Arnaud Sangnier and Jan Stückrath. *On the Decidability Status of Reachability and Coverability in Graph Transformation Systems.* In *Proceedings of the 23rd International Conference on Rewriting Techniques and Applications (RTA'12)*, LIPIcs 15, pages 101-116. Leibniz-Zentrum fur Informatik, 2012.

[Abd+11] Parosh Abdulla, Giorgio Delzanno and Othmane Rezine and Arnaud Sangnier and Riccardo Traverso. *On the Verification of Timed Ad Hoc Networks.* In *Proceedings of the 9th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'11)*, LNCS 6919, pages 256-270. Springer, 2011.

[DSZ11] Giorgio Delzanno, Gianluigi Zavattaro and Arnaud Sangnier. *On the Power of Cliques in the Parameterized Verification of Ad Hoc Networks.* In *Proceedings of the 14th International Conference on Foundations of Software Science and Computation Structures (FOSSACS'11)*, LNCS 6269, pages 313-327. Springer, 2011.

[LS10] Sébastien Labbé and Arnaud Sangnier. *Formal verification of industrial software with dynamic memory management.* In *Proceedings of the 16th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC'10)*, pages 77-84, IEEE Computer Society, 2010.

[DSZ10] Giorgio Delzanno, Gianluigi Zavattaro and Arnaud Sangnier. *Parameterized Verification of Ad Hoc Networks.* In *Proceedings of the 21st International Conference on Concurrency Theory (CONCUR'10)*, LNCS 6269, pages 313-327. Springer, 2010.

[DS10] Stéphane Demri and Arnaud Sangnier. *When Model-Checking Freeze LTL over Counter Machines Becomes Decidable.* In *Proceedings of the 13th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'10)*, LNCS 6014, pages 176-190. Springer, 2010.

[FS10] Alain Finkel and Arnaud Sangnier. *Mixing coverability and reachability to analyze VASS with one zero-test.* In *Proceedings of the 36th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'10)*, LNCS 5901, pages 394-406. Springer, 2010.

[RS09] Pierre-Alain Reynier and Arnaud Sangnier. *Weak Time Petri Nets strike back!* In *Proceedings of the 20th International Conference on Concurrency Theory (CONCUR'09)*, LNCS 5710, pages 557-571. Springer, 2009.

[FLS07] Alain Finkel, Etienne Lozes and Arnaud Sangnier. *Towards model-checking programs with lists.* In *Infinity in Logic and Computation, Selected Papers of the International Conference held in Cape Town, South Africa, November 2007*, LNAI 5489, pages 56-86. Springer, 2009.

[FS08] Alain Finkel and Arnaud Sangnier. *Reversal-bounded counter machines revisited.* In *Proceedings of the 33rd International Symposium on Mathematical Fundations of Computer Science (MFCS'08)*, LNCS 5162, pages 323-334. Springer, 2008.

[DLS08] Stéphane Demri, Ranko Lazić and Arnaud Sangnier. *Model checking freeze LTL over one-counter automata.* In *Proceedings of the 11th International Conference on Foundations of Software Science and Computation Structures (FOSSACS'08)*, LNCS 4962, pages 490-504. Springer, 2008.

[D'A+07] Davide D'Aprile, Susanna Donatelli, Arnaud Sangnier and Jeremy Sproston. *From Time Petri Nets to Timed Automata: an Untimed Approach.* In *Proceedings of the 13th International Conference on Tools and Algorithms for Constructions and Analysis of Systems (TACAS'07)*, LNCS 4424, pages 216-230. Springer, 2007.

1.3 SUPERVISIONS

Since I have defended my PhD, I have (co)-supervised the following students:

- PhD of Olivier Stietel
  - *Period:* 2019-2023
  - *Subject:* Formal tools for the design of distributed algorithms
  - *Co-supervisor:* Benedikt Bollig
  - *Related publications:* [BSS21]
- PhD of Paulin Fournier
  - *Period:* 2012-2015
  - *Subject:* Parameterised Verification of probabilistic systems
  - *Co-supervisors:* Nathalie Bertrand and Thierry Jéron
  - *Related publications:* [BFS14; BFS15]
- PhD of Amit Kumar Dhar
  - *Period:* 2011-2014
  - *Subject:* Algorithms for Model-checking Flat Counter Systems
  - *Co-supervisor:* Stéphane Demri
  - *Related publications:* [DDS18; DDS15; DDS14; DDS13; DDS12]
- Master 2 thesis of Lucie Guillou
  - *Period:* 2022
  - *Subject:* Verification of parameterised networks
  - *Co-supervisor:* Nathalie Sznajder
- Master 2 thesis of Loriane Leclerq
  - *Period:* 2021
  - *Subject:* QLTL : expressivity and complexity with the 'structure' semantics

  - *Co-supervisor:* François Laroussinie

- Master 2 thesis of Fedor Ryabinin
  - *Period:* 2018
  - *Subject:* Formal Methods for the Development of Distributed Algorithms
  - *Co-supervisor:* Benedikt Bollig
  - *Related publications:* [BRS21]

- Master 2 thesis of Amit Kumar Dhar
  - *Period:* 2011
  - *Subject:* Counter Systems with Presburger-definable Reachability Sets: Decidability and Complexity
  - *Co-supervisor:* Stéphane Demri

- Internship of Julien Roupin
  - *Period:* 2021
  - *Subject:* Verification of parameterised systems with counters

2 MATHEMATICAL TOOLS

In this chapter, we will define properly the different mathematical tools we use along the thesis.

2.1 GENERAL NOTATIONS

We denote by $\mathbb{N}$, $\mathbb{Z}$ and $\mathbb{R}$ the set of natural, integer and real numbers, respectively. For $i, j \in \mathbb{Z}$ with $i \leq j$, we use $[i, j]$ to represent the set $\{k \in \mathbb{Z} \mid i \leq k \text{ and } k \leq j\}$. We denote the infinity by the symbol ∞ and we assume that $i < \infty$ for all $i \in \mathbb{Z}$.

For a natural $n \geq 1$ and a set of elements E, we use E^n to denote the set of vectors of n-elements, also called n-dim vectors of E. For $\mathbf{v} \in E^n$, the i-th elements of $\mathbf{v}$ is $\mathbf{v}[i]$ for every $i \in [1, n]$.

Given two naturals $n, m \geq 1$ and a set E, the set $E^{n \times m}$ corresponds to the matrices with n rows and m columns with values in E. For $\mathbf{A} \in E^{n \times m}$, we use $\mathbf{A}[i]$ to represent te i-th column of $\mathbf{A}$ which belongs to E^m and $\mathbf{A}[i][j]$ is the element of E corresponding to the the entry of $\mathbf{A}$ on the i-th row and the j-th colum for each $i \in [1, n]$ and each $j \in [1, m]$. We assume that the sets $E^{n \times 1}$ and E^n are equal. The set $E^{n \times n}$ is the set of *square matrices* of dimension n over E. We denote by $\mathbf{I}_n$ the *identity matrix* in $\mathbb{Z}^{n \times n}$. Given two matrices $\mathbf{A} \in \mathbb{Z}^{n \times m}$ and $\mathbf{B} \in \mathbb{Z}^{m \times p}$, we denote by $\mathbf{A} \cdot \mathbf{B}$ the matrix product of $\mathbf{A}$ and $\mathbf{B}$ which belongs to $\mathbb{Z}^{n \times p}$. For a square matrix $\mathbf{A} \in \mathbb{Z}^{n \times n}$, we define $\mathbf{A}^0 = \mathbf{I}_n$ and $\mathbf{A}^i = \mathbf{A}^{i-1} \cdot \mathbf{A}$ for all naturals $i > 0$.

An *affine function* $f : \mathbb{Z}^n \times \mathbb{Z}^n$ is defined by a pair $(\mathbf{A}, \mathbf{b})$ with $\mathbf{A} \in \mathbb{Z}^{n \times n}$ and $\mathbf{b} \in \mathbb{Z}^n$ and is such that for all $\mathbf{v} \in \mathbb{Z}^n$, we have $f(\mathbf{v}) = \mathbf{A} \cdot \mathbf{v} + \mathbf{b}$. We identify an affine function f with its definition $(\mathbf{A}, \mathbf{b})$ and denote by Aff_n the set of affine functions over $\mathbb{Z}^n$. A *translation* is an affiche fonction $f \in \mathsf{Aff}_n$ such that $f = (\mathbf{I}_n, \mathbf{b})$. We identify a translation simply by its vector $\mathbf{b}$. We denote by Trsl_n the set of translations over $\mathbb{Z}^n$.

Except when we will specify it explicitely, we assume that the integers used in our formalisms are encoded in binary. Hence the size of an integer $a \in \mathbb{Z}$ is $\mathtt{size}(a) = \log_2(|a|)$ (where $|a|$ denotes the absolute value of a), the size of a n-dim vector of integers $\mathbf{b} \in \mathbb{Z}^n$ is $\mathtt{size}(\mathbf{b}) = \sum_{i=1}^{n} \log_2(|b[i]|)$ and for a matrix $\mathbf{A} \in \mathbb{Z}^{n \times m}$, we have $\mathtt{size}(\mathbf{A}) = \sum_{i=1}^{n} \sum_{j=1}^{m} \log_2(|\mathbf{A}[i][j]|)$.

A probability distribution on a countable set X is a function $f : X \mapsto \{p \in \mathbb{R} \mid 0 \leq 0 \leq 1\}$ such that $\Sigma_{x \in X} f(x) = 1$. We use $\mathtt{Dist}(X)$ to denote the set of all probability distributions on X.

For a finite set of elements E, its cardinal is $card(E)$.

2.2 MULTISETS

For a finite set E, the set $\mathbb{N}^E$ represents the multisets over E. For two elements $M, M' \in \mathbb{N}^E$, we denote by $M + M'$ the multiset such that $(M + M')(e) = M(e) + M'(e)$ for all $e \in E$. We say that $M \leq M'$ if and only if $M(e) \leq M'(e)$ for all $e \in E$. If $M \leq M'$, then $M' - M$ is the multiset such that $(M' - M)(e) = M'(e) - M(e)$ for all $e \in E$. The size of a multiset M is given by $\mathtt{size}(M) = \Sigma_{e \in E} \log_2(M(e))$. For $e \in E$, we use sometimes the notation e

for the multiset M verifying $M(e) = 1$ and $M(e') = 0$ for all $e' \in E \setminus \{e\}$ and the notation $\langle\langle e1, e1, e2, e3\rangle\rangle$ to represent the multiset with four elements $e1, e1, e2$ and $e3$.

2.3 FINITE AND INFINITE WORDS

For a finite alphabet Σ, the set of finite words over Σ is denoted by Σ^* and we use Σ^ω for the set of infinite words (or ω-words) over Σ. For a finite word $w = a_1 \dots a_k$ over Σ, we write $\texttt{len}(w)$ to denote its *length* k. For every $0 \leq i < \texttt{len}(w)$, the $(i+1)$-th letter of the word w is denoted by $w(i)$, here equals to a_{i+1}. For $i, j \in [0, \texttt{len}(w) - 1]$ such that $i \leq j$, the subword $a_{i+1} \dots a_{j+1}$ of w is denoted by $w[i, j]$. By convention, for an infinite word $w \in \Sigma^\omega$, we assume that $\texttt{len}(w) = \infty$ and we adopt the same notations $w(i)$ and $w[i, j]$ as for finite words.

2.4 PRESBURGER ARITHMETIC AND SEMI-LINEAR SETS

In order to represent finitely infinite sets of naturals, it is useful to have a logical language with nice closure and decidability properties. Presburger arithmetic responds exactly to these requirements and we shall recall here the definition and some properties of this language. Presburger arithmetic corresponds to the first order logic over the structure $(\mathbb{N}, 0, 1, +, <)$. In order to ease the representation, we define the syntax of Presburger formulas over a set of variables X as follows:

$$\phi ::= \sum_{1 \leq i \leq n} a_i \cdot x_i \sim b \mid \exists x.\phi \mid \neg\phi \mid \phi \vee \phi \mid \phi \wedge \phi$$

where $\{x_1, \dots, x_n, x\} \subseteq X$, $a_1, \dots, a_n \in \mathbb{Z}$, $b \in \mathbb{N}$ and $\sim \in \{<, \leq, =, >, \geq\}$. A quantifier free Presburger formula is obtained by removing from the syntax the existential quantification $\exists x.\phi$. The set of free variables of a formula ϕ is defined in the usual way and denoted by $var(\phi)$. We will sometimes use the notation $\phi(x_1, \dots, x_n)$ to specify that the free variables of ϕ are included in $x_1, \dots, x_n$. When clear from the context, this set of variables might be omitted.

For a natural $n \geq 1$, a vector $\mathbf{v} \in \mathbb{N}^n$ satisfies a Presburger formula $\phi(x_1, \dots, x_n)$ if and only if the first order formula obtained by replacing each x_i by $\mathbf{v}[i]$ is valid over the naturals in the classical sense. If this is the case we write $\mathbf{v} \models \phi(x_1, \dots, x_n)$. We use the notation $[\![\phi(x_1, \dots, x_n)]\!]$ to represent the set $\{\mathbf{v} \in \mathbb{N}^n \mid \mathbf{v} \models \phi(x_1, \dots, x_n)\}$.

Since we will present some complexity results, it is important to state properly what is the size of a Presburger formula ϕ. For this matter, we define inductively the function $\texttt{size}(\cdot)$ as follows: $\texttt{size}(\sum_{1 \leq i \leq n} a_i \cdot x_i \sim b) = n + \sum_{1 \leq i \leq n} \log_2(|a_i|) + \log_2(b)$, $\texttt{size}(\exists x.\phi) = 1 + \texttt{size}(\phi)$, $\texttt{size}(\neg\phi) = 1 + \texttt{size}(\phi)$, $\texttt{size}(\phi_1 \vee \phi_2) = 1 + \texttt{size}(\phi_1) + \texttt{size}(\phi_2)$ and $\texttt{size}(\phi_1 \wedge \phi_2) = 1 + \texttt{size}(\phi_1) + \texttt{size}(\phi_2)$.

Remark. *We assume that the naturals used in Presburger formulas are encoded in binary but using an unary encoding or a binary encoding does not change the complexity of the validity problem as explained for instance in [Haa14, p. 3]. The reason being that binary encoding can be encoding into unary encoding using extra existentially quantified variables and multiplication by two leading to a subquadratic blowup in the formula size .*

By definition, Presburger arithmetic enjoys nice logical closure properties and furthermore Mojżesz Presburger shows that it has a decidable satisfiability problem. The satisfiability

problem consists in determining given a close formula ϕ (i.e. without free variable) whether it is valid.

Theorem 2.1. *[Pre29] The satisfiability problem for Presbruger arithmetic is decidable.*

Furthermore thanks to a study of the complexity for this problem performed in [Ber77] we know that the satisfiability problem for Presburger arithmetic is 2EXPTIME-hard and in 2EXPSPACE, hence as well in 3EXPTIME. This result is recalled as well in [Haa14], where a detailed analysis of the complexity of the satisfiability for Presburger arithmetic according to number of alternations of quantifiers is performed. We will need in this thesis another complexity result for the satisfiability of quantifier free Presburger formulas. In that case the satisfiability problem asks given a quantifier free formula $\phi(x_1, \ldots, x_n)$ (with at least one variable) whether $\exists x_1. \ldots \exists x_n. \phi$ is valid. Considering quantifier free formulas allows to lower significantly the complexity of the satisfiability problem.

Theorem 2.2. *[BT76] The satisfiability problem for quantifier free Presbruger arithmetic is* NP*-complete.*

We say that a subset of E of $\mathbb{N}^n$ (for $n \geq 1$) is Presburger definable if and only if there exists a Presburger formula $\phi_E(x_1, \ldots, x_n)$ such that $E = [\![\phi_E(x_1, \ldots, x_n)]\!]$.We will need as well to express when a relation over vectors of naturals is expressible in Presburger arithmetic. Given two naturals $n, m \geq 1$, we say that a relation $R \subseteq \mathbb{N}^n \times \mathbb{N}^m$ is Presburger definable if and only if there exists a Presburger formula $\phi_R(x_1, \ldots, x_n, y_1, \ldots, y_m)$ such that for all $\mathbf{v} \in \mathbb{N}^n$ and $\mathbf{w} \in \mathbb{N}^m$ we have $(\mathbf{v}, \mathbf{w}) \in R$ if and only if $\mathbf{v}, \mathbf{w} \models \phi_R(x_1, \ldots, x_n, y_1, \ldots, y_m)$ (where we consider the pair $(\mathbf{v}, \mathbf{w})$ as a unique vector in $\mathbb{N}^{n+m}$ such that $(\mathbf{v}, \mathbf{w})[i] = \mathbf{v}[i]$ for all $i \in [1, n]$ and $(\mathbf{v}, \mathbf{w})[n + i] = \mathbf{w}[i]$ for all $i \in [1, m]$).

2.5 WELL-QUASI-ORDERS

A *quasi-order* is defined by a pair $(E, \leq)$ where E is a set of elements and $\leq \subseteq E \times E$ is a reflexive and transitive relation. A quasi-order $(E, \leq)$ is a *well-quasi-order* (wqo) if and only if for any infinite sequence $(e_i)_{i \in \mathbb{N}}$ of elements of E, there exist two indices $i, j \in \mathbb{N}$ such that $e_i \leq e_j$.

Assume $(E, \leq)$ is a quasi-order. A set $U \subseteq E$ is said to be *upward-closed* if and only if $e \in U$ and $e \leq e'$ implies $e' \in U$. Given a subset of elements $D \subseteq E$, the upward closure of D, denoted by $\uparrow D$, is the set of elements $\{e \in E \mid \exists d \in D. d \leq e\}$. Consequently, an upward-closed set is a set U such that $\uparrow U = U$. If $(E, \leq)$ is a wqo, then upward-closed sets can be represented in a finite matter, thanks to a finite *basis*, as stated by the following lemma.

Lemma 2.1. *[Hig52] If $(E, \leq)$ is a wqo then for all upward-closed set $U \subseteq E$ there exists a finite subset $B \subseteq E$ such that $\uparrow B = U$.*

From the definition of wqo, we can furthermore obtain another result concerning the increasing sequences (with respect to inclusion) of upward-closed sets (see e.g. [FS01] for a proof).

Lemma 2.2. *Let $(E, \leq)$ be a wqo. If $(I_i)_{i \in \mathbb{N}}$ is a sequence of upward-closed sets such that $I_i \subseteq I_{i+1}$ for all $i \in \mathbb{N}$, then there exists $k \in \mathbb{N}$ such that $I_i = I_k$ for all $i \geq k$.*

Part I

VERIFICATION OF COUNTER SYSTEMS

In this part, I will detail the results I obtained on the verification of infinite-state systems manipulating natural variables, also known as counter systems. Since simple verification problems are undecidable for such systems due to the fact that one can simulate the tape of a Turing machine with two counters [Min67], I have studied restrictions of these models for which verification problems are decidable. I hence present here different decidability and complexity results for the model-checking of restrictions of counter systems considering different families of specification languages.

3

TAXONOMY OF COUNTER SYSTEMS

In this chapter, we introduce the model of counter systems which are basically finite state machines equipped with a finite set of variables having natural values.

3.1 GENERAL MODEL

A counter system has access to various natural variables, called the counters, and it can test their value and update them. Once that said it opens a wide world of mathematical opportunities to define such tests and updates. In the systems we consider, the tests, also called guards, are quantifier-free Presburger formulas and the updates are affine fonctions. We begin by providing formal definitions of these mechanisms.

Let $\mathtt{C} = \{\mathtt{x}_1, \mathtt{x}_2, \ldots\}$ be a countably infinite set of counters (variables interpreted over non-negative integers) and $\mathrm{A}T = \{\mathtt{p}_1, \mathtt{p}_2, \ldots\}$ be a countably infinite set of propositional variables (abstract properties about program points). We write $\mathtt{C}_n$ to denote the restriction of $\mathtt{C}$ to $\{\mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{x}_n\}$. A *guard* $\mathtt{g}$ using the counters from $\mathtt{C}_n$, written $\mathtt{G}(\mathtt{C}_n)$, is made of Boolean combinations of atomic guards of the form $\sum_{i=1}^{n} a_i \cdot \mathtt{x}_i \sim b$ where the a_i's are in $\mathbb{Z}$, $b \in \mathbb{N}$ and $\sim \in \{=, \leq, \geq, <, >\}$. Given a guard $\mathtt{g} \in \mathtt{G}(\mathtt{C}_n)$ and a vector $\mathbf{v} \in \mathbb{N}^n$, we say that $\mathbf{v}$ satisfies $\mathtt{g}$, written $\mathbf{v} \models \mathtt{g}$, if the formula obtained by replacing each $\mathtt{x}_i$ by $\mathbf{v}[i]$ holds. We denote by $\top$ the guard satisfied by all vectors in $\mathbb{N}^n$ (it is equivalent to $\mathtt{x}_1 \geq 0$).

Definition 3.1 (Affine Counter System)**.** *For a natural number $n \geq 1$, an* Affine Counter System *S of dimension n is a tuple $(Q, \mathtt{C}_n, \Delta, \mathbf{l})$ where:*

- *Q is a finite set of control states,*
- *$\mathbf{l} : Q \to 2^{\mathrm{A}T}$ is a labeling function,*
- *$\Delta \subseteq Q \times \mathtt{G}(\mathtt{C}_n) \times \mathsf{Aff}_n \times Q$ is a* finite *set of edges labelled by guards and affine functions to update the counter values.*

For $\delta = (q, \mathtt{g}, (\mathbf{A}, \mathbf{b}), q')$ in Δ, we use the following notations: $\mathit{source}(\delta) = q$, $\mathit{target}(\delta) = q'$, $\mathit{guard}(\delta) = \mathtt{g}$ and $\mathit{update}(\delta) = (\mathbf{A}, \mathbf{b})$. The size of a counter system $S = (Q, \mathtt{C}_n, \Delta, \mathbf{l})$ is equal to $\mathtt{size}(S) = \sum_{\delta \in \Delta} \mathtt{size}(\delta) + \sum_{q \in Q} \mathit{card}(\mathbf{l}(q))$ where $\mathtt{size}(\delta) = 1 + \mathtt{size}(\mathit{guard}(\delta)) + \mathtt{size}(\mathit{update}(\delta))$. We denote by $\mathcal{ACS}$ the class of Affine Counter Systems.

As usual, to an Affine Counter System $S = (Q, \mathtt{C}_n, \Delta, \mathbf{l})$, we associate a transition system $\mathfrak{T}(S) = (Q \times \mathbb{N}^n, \to)$ where $Q \times \mathbb{N}^n$ is the set of configurations and $\to \subseteq (Q \times \mathbb{N}^n) \times \Delta \times (Q \times \mathbb{N}^n)$ is the transition relation defined by: $((q, \mathbf{v}), \delta, (q', \mathbf{v}')) \in \to$ (also written $(q, \mathbf{v}) \xrightarrow{\delta} (q', \mathbf{v}')$) if and only if the conditions below are satisfied:

- $q = \mathit{source}(\delta)$ and $q' = \mathit{target}(\delta)$,
- $\mathbf{v} \models \mathit{guard}(\delta)$ and if $\mathit{update}(\delta) = (\mathbf{A}, \mathbf{b})$ then $\mathbf{v}' = \mathbf{A} \cdot \mathbf{v} + \mathbf{b}$.

Note that in such a transition system, the counter values are non-negative. We write $(q, \mathbf{v}) \to (q', \mathbf{v}')$ if there exists $\delta \in \Delta$ such that $(q, \mathbf{v}) \xrightarrow{\delta} (q', \mathbf{v}')$ and denote by $\to^*$ the reflexive and transitive closure of $\to$.

Given an initial configuration $c_0 \in Q \times \mathbb{N}^n$, an infinite run (or execution) ρ starting from c_0 in S is an infinite path in the associated transition system $\mathfrak{T}(S)$ denoted as:

$$\rho := c_0 \xrightarrow{\delta_0} \cdots \xrightarrow{\delta_{m-1}} c_m \xrightarrow{\delta_m} \cdots$$

where $c_i \in Q \times \mathbb{N}^n$ and $\delta_i \in \Delta$ for all $i \in \mathbb{N}$. A finite run (or execution) is defined similarly by considering finite paths in $\mathfrak{T}(S)$.

```
1  int x = 10;
2  int y = 0;
3  int z = 0;
4  while (x > 0) {
5    y = 1;
6    while (y < 10) {
7      y = 2 * y;
8    }
9    z = z + y;
10   x=x-1;
11 }
```

Figure 3.1: A simple C program manipulating integer variables

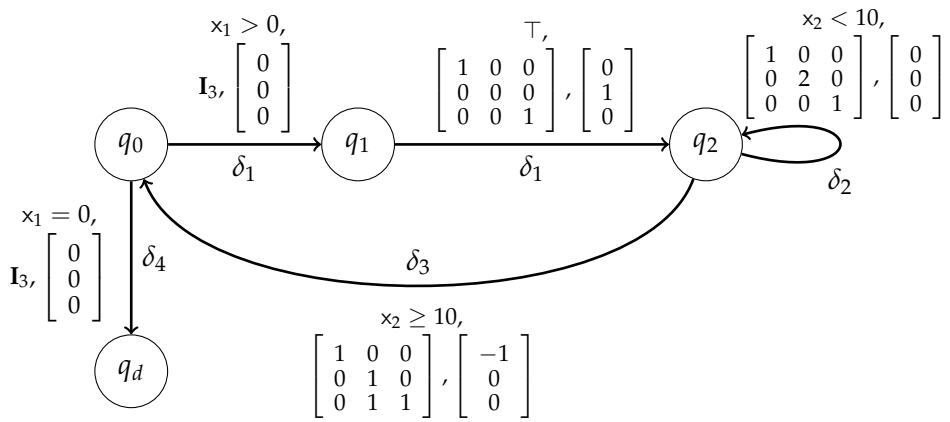

Figure 3.2: An Affine Counter System for the program of Figure 3.1

Example 3.1. *In Figure 3.2, we give an example of an affine counter system (where we have omitted the atomic propositions) and which corresponds to an encoding of the C program presented in Figure 3.1. In the system, the counter* x_1 *encodes the variable* `x`, x_2 *the variable* `y` *and* x_3 *the variable* `z` *of the program. The while loop of Line 6 is encoded in a single looping transition on the state* q_2 *and the transition that goes out of this loop together with the instructions of Line 9 and 10 are encoded as well in a single transition from* q_2 *to* q_0*, Of course, we could have encoded this program differently, for instance by performing the loop of Line 6 with two transitions, one for the test and one for the instruction* `y=2*y` *and the way the encoding is performed strongly depends on the properties one wishes to verify on the program under analysis. Note as well that we did not encode in the system*

the initial values of the variables as we assume that their will be provided by specifying some initial configuration. A finite run of the system shown in Figure 3.2 is then for instance :

$$(q_0, \begin{bmatrix} 10 \\ 0 \\ 0 \end{bmatrix}) \xrightarrow{\delta_0} (q_1, \begin{bmatrix} 10 \\ 0 \\ 0 \end{bmatrix}) \xrightarrow{\delta_1} (q_2, \begin{bmatrix} 10 \\ 1 \\ 0 \end{bmatrix}) \xrightarrow{\delta_2} (q_2, \begin{bmatrix} 10 \\ 2 \\ 0 \end{bmatrix}) \xrightarrow{\delta_2} (q_2, \begin{bmatrix} 10 \\ 4 \\ 0 \end{bmatrix}) \xrightarrow{\delta_2}$$
$$(q_2, \begin{bmatrix} 10 \\ 8 \\ 0 \end{bmatrix}) \xrightarrow{\delta_2} (q_2, \begin{bmatrix} 10 \\ 16 \\ 0 \end{bmatrix}) \xrightarrow{\delta_3} (q_0, \begin{bmatrix} 9 \\ 16 \\ 16 \end{bmatrix})$$

3.2 REACHABILITY PROBLEMS

When one wants to verify whether the behavior of a system is correct or not, one key problem is the so-called reachability problem which consists in asking whether a certain state is reachable in the representation of the system. Most of the time such a state represents an error state (or a set of error states), and if it cannot be reached it means that the system under verification is safe. In some cases, such a reachability property can as well be used to check other behavior of the system. For instance in a finite automaton, if an accepting state is reachable from an initial state then the automaton recognizes a non empty language. In a graph, the reachability problem asks given two vertices s and t whether there is a path from s to t and it is well known that this problem is NL-complete (see for instance [Pap94]).

For the case of Affine Counter Systems, we can define two variants of this problem. The first one asks only if a control state is reachable from a given initial configuration and is defined formally as follows:

CS-CONTROLREACH

Input: An Affine Counter System $S = (Q, \mathtt{C}_n, \Delta, \mathbf{l})$, an initial configuration $(q_0, \mathbf{v}_0) \in Q \times \mathbb{N}^n$ and a control state $q_f \in Q$;

Question: Does there exist $\mathbf{v} \in \mathbb{N}^n$ such that $(q_0, \mathbf{v}_0) \rightarrow^* (q_f, \mathbf{v})$?

The other reachability problem asks for the reachability of a complete configuration.

CS-REACH

Input: An Affine Counter System $S = (Q, \mathtt{C}_n, \Delta, \mathbf{l})$, an initial configuration $(q_0, \mathbf{v}_0) \in Q \times \mathbb{N}^n$ and a configuration $(q_f, \mathbf{v}_f) \in Q \times \mathbb{N}^n$;

Question: Does $(q_0, \mathbf{v}_0) \rightarrow^* (q_f, \mathbf{v}_f)$ hold?

Note that for Affine Counter Systems in their full generality, there is no need to make a distinction between CS-CONTROLREACH and CS-REACH since CS-REACH can be reduced to CS-CONTROLREACH by adding a transition to a specific extra state which tests whether the values of the n counters are equal to $\mathbf{v}_f$. However we shall see later in this chapter that in some cases, as for instance when considering Vector Addition System with States, the distinction between these two problems does make sense.

In [Min67], Minsky introduces a specific class of programs manipulating variables over the naturals and shows that it was possible to encode the behavior of Türing machines in such programs. We call a *deterministic Minsky machine* a program which manipulates two natural variables (or counters) x_1 and x_2, and which is composed of a finite set of instructions. Each of the instruction is either of the form (1) $L : x_i := x_i + 1;$ `goto` L' or (2) $L :$ `if` $x_i = 0$ `then goto` L' `else` $x_i := x_i - 1;$ `goto` L'' where $i \in \{1, 2\}$ and L, L', L'' are labels preceding each instruction. Furthermore there is a special label L_F from which nothing can be done. The behavior of each instruction is then the expected one. Minsky proves that the *halting problem* for deterministic Minsky machines which consists in deciding whether the execution that starts from L_0 with counters equal to 0 reaches L_F is undecidable. In our context, we see that deterministic Minsky machines can be seen as a specific case of Affine Counter Systems and hence from Minsky's result, one easily deduces the following theorem.

Theorem 3.1. CS-ControlReach *and* CS-Reach *are undecidable.*

3.3 VARIOUS RESTRICTIONS

Even though the undecidability results stated in Theorem 3.1 leave few hopes for the verification of counter systems, it happens that decidability can be regained by imposing some syntactic or semantic restrictions on this model. We list here some of the restricted classes of counter systems we will use in this work and recall how they behave with respect to the reachability problems.

3.3.1 *Kripke Structures*

First, even if Kripke Structures are a model independant of counter systems, in order to reuse the notations, we can see a Kripke Structure as an Affine Counter System without counter.

Definition 3.2 (Kripke Structure)**.** *A* Kripke Structure *S n is a tuple $(Q, \Delta, \mathbf{l})$ where:*

- *Q is a finite set of control states.*
- *$\mathbf{l} : Q \to 2^{AT}$ is a labeling function.*
- *$\Delta \subseteq Q \times Q$ is a* finite *set of edges.*

Since there is no counter value, configurations of Kripke Structures will simply be the set of control states and the transition relation $\to$ will be equals to Δ. However we transpose in the obvious way the notions of runs to Kripke Structures. We denote by $\mathcal{KS}$ the class of Kripke Structures.

Since Kripke Structures are finite states models, they enjoy many decidable properties, but we will see that in some specific cases it is not harder to verify some counter systems than it is to verify Kripke Structures. Furthermore from the reachability problem in graph, we obtain automatically the following result (see for instance [Pap94])..

Theorem 3.2. CS-Reach *for Kripke Structures is* NL*-complete.*

3.3.2 *Translating Counter Systems*

Another restriction that can be made concerns the updates performed by the system. Instead of allowing the full class of affine functions, in Translating Counter Systems we restrict ourselves to translations. Hence an Affine Counter System $S = (Q, \mathtt{C}_n, \Delta, \mathbf{l})$ of dimension n is a *Translating Counter System* if and only if for all $\delta \in \Delta$ we have $update(\delta) = (\mathbf{I}_n, \mathbf{b})$ (where $\mathbf{I}_n$ is the identity matrix). In that case, when it is clear that we consider a Translating Counter System, we assume that $\Delta \subseteq Q \times \mathtt{G}(\mathtt{C}_n) \times \mathbb{Z}^n \times Q$ and for each edge $\delta \in \Delta$ we have $update(\delta) = \mathbf{b}$. The undecidability results of Theorem 3.1 still hold for Translating Counter Systems as in Minsky machines, the update operation of counters, which are simple decrement or increment, are clearly some translations. We denote by $\mathcal{TCS}$ the class of Translating Counter Systems.

3.3.3 *One Counter Systems*

As we have seen, the undecidability result for the reachability problems in counter systems is a consequence of the undecidability of the halting problem for Minsky machines, which manipulate two natural variables. But if we consider systems with a single counter, such a reduction does not hold anymore and in fact we shall see that many verification problems become decidable under this restriction. In this work, we call a *One Counter System* a Translating Counter System of dimension 1. Since these systems manipulate a single counter we can simplify the shape of the guards and we hence assume that there all of the form $x_1 \sim b$ with $b \in \mathbb{N}$ and $\sim \in \{=, \leq, \geq, <, >\}$. We denote by $\mathcal{OCS}$ the class of One Counter Systems.

We can impose a further restriction on One Counter Systems by allowing a single form of guards which can only test whether the counter is equal to 0 (we call such guards zero test) or not. A One Counter System $S = (Q, \mathtt{C}_1, \Delta, \mathbf{l})$ is said to be *simple* if and only if $guard(\delta) = x_1 = 0$ or $guard(\delta) = true$ for each edge $\delta \in \Delta$. Such a restriction can be explained by different facts. First zero tests give the ability to test equality for any other values as we explain later and if one take Translating Counter Systems with no guards (see the next part on Vector Addition System with States), adding zero tests is what make the verification undecidable in many cases. Second One Counter Systems can be seen as a restriction of pushdown systems with a single stack symbol and using zero tests allow these systems to check whether the bottom of the stack is reached. Hence zero tests are an important feature in Counter Systems and as we explain in the following remarks, even if any One Counter System can be simulated by a simple One Counter System, the simulation might have a cost.

Remark. *The semantics of counter systems allows to test easily whether the value of a counter is strictly positive without using any guard. To do so one can use two consecutive transitions, one that decrement by* 1 *the counter and the next one that restore the value thanks to an increment by* 1*. Since the counter values are naturals, the system will go through these two transitions if and only if its counter value is strictly positive. This is why in Simple One Counter Systems, even though there is no guard of the form* $x_1 > 0$*, those systems are able to perform such a test.*

Remark. *One can simulate a One Counter System with a Simple One Counter System but in some cases this simulation might have a cost. The test of the form* $x_1 > k$ *can be simulated by a transition that decrements the counter by* $k + 1$ *and a consecutive transition which restores the counter value.*

A similar trick works for test $x_1 \geq k$ or $x_1 = k$. However to simulate the test $x_1 < k$, one option is to replace it by $k-1$ transitions each one testing whether the counter value is equal to i for i in $[0, k-1]$, however since k is encoded in binary, this translation might lead to an explosion blowup.

One nice result about Simple One Counter Systems is that even if they are equipped with a counting mechanisms, their reachability problems are not harder that in simple graphs when the updates are encoded in unary. This result is obtained by proving that to reach a control state, there is no need for the counter to take very high value.

Theorem 3.3. *[LLT05; Chi+19]* CS-ControlReach *and* CS-Reach *for Simple One Counter Systems where the initial and final configurations and the updates are encoded in unary are* NL*-complete.*

If we assume a binary encoding, then using the previous result would lead to a Pspace-algorithm, however as shown in [Haa+09], it is possible to get a better complexity result. The proof technique here is a bit difference and is based on the existence of a reachability certificate of polynomial size that one can then guess.

Theorem 3.4. *[Haa+09]* CS-ControlReach *and* CS-Reach *for Simple One Counter Systems are* NP*-complete.*

3.3.4 *Vector Addition Systems with States and Petri nets*

One other way to divert the undecidability result on Minsky machines consists in prohibiting the system to test the values of the counters and if we consider only translations we obtain he class of Vector Addition System with States. As a matter of fact a Translating Counter System $S = (Q, C_n, \Delta, \mathbf{1})$ of dimension n is a *Vector Addition System with States* (VASS) if and only if for all $\delta \in \Delta$ we have $guard(\delta) = true$. As we have seen in the case of Simple One Counter Systems, it is not really true that the system cannot test the counter value, in fact he has only the ability to test whether some counters are greater than some constant (by performing a decrement followed by an increment). We denote by $\mathcal{VASS}$ the class of VASS. This class is furthermore very interesting because of its promiscuity with the model of Petri nets.

Petri nets were introduced by C.A. Petri in its PhD thesis [Pet62] as a model for concurrent systems. This model has then received a lot of attention both for practical applications and from the community of researchers in formal model. In fact, the rules for this model are quite simple but allow nevertheless to characterize the behavior of many concurrent systems. From the theoretical point of view, the various algorithms designed for their analysis have led to the development of many mathematical tools that have happened to be useful for the verification of other classes of systems. We now present more in detail this model.

Definition 3.3 (Petri Net)**.** *A Petri Net N is a tuple $(P, T, Pre, Post)$ where:*

- *P is a finite set of places,*
- *T is a finite set of transitions,*
- *$Pre : T \mapsto \mathbb{N}^P$ is the precondition function,*
- *$Post : T \mapsto \mathbb{N}^P$ is the postcondition function.*

A marking of a Petri Net is a multiset $M \in \mathbb{N}^P$. A Petri Net defines a transition relation $\Rightarrow \subseteq \mathbb{N}^P \times T \times \mathbb{N}^P$ such that $M \overset{t}{\Rightarrow} M'$ for $M, M' \in \mathbb{N}^P$ and $t \in T$ if and only if $M \geq Pre(t)$ and $M' = M - Pre(t) + Post(t)$. The intuition behinds Petri Nets is that marking put tokens in some places and each transition consumes with *Pre* some tokens and produces others thanks to *Post* in order to create a new marking. We write $M \Rightarrow M'$ if and only if there exists $t \in T$ such that $M \overset{t}{\Rightarrow} M'$ and denote by $\Rightarrow^*$ the reflexive and transitive closure of $\Rightarrow$. A finite run (or execution) is then a finite sequence of the form $M_0 \overset{t_0}{\Rightarrow} M_1 \overset{t_1}{\Rightarrow} M_2 \overset{t_2}{\Rightarrow} \dots \overset{t_{k-1}}{\Rightarrow} M_k$. Infinite runs (or executions) are defined similarly.

As for counter systems, we present two classical problems on Petri Nets. The first one is similar to CS-ControlReach, the difference being that in Petri Nets we do not have control states, it is known as the coverability problem and it asks whether there is a reachable marking that covers (is greater than) a given marking. It is similar to CS-ControlReach because here as well the precise value of the reached marking does not matter.

PN-Coverability

Input: A Petri Net $N = (P, T, Pre, Post)$,
an initial marking $M_0 \in \mathbb{N}^P$,
and a marking $M \in \mathbb{N}^P$;

Question: Does there exist $M' \in \mathbb{N}^P$ such that $M_0 \Rightarrow^* M'$ and $M \leq M'$?

The other problem is the version where we ask to reach the given marking, it is known as the reachability problem for Petri Nets.

PN-Reach

Input: A Petri net $N = (P, T, Pre, Post)$,
an initial marking $M_0 \in \mathbb{N}^P$,
and a marking $M \in \mathbb{N}^P$;

Question: Does there exist $M' \in \mathbb{N}^P$ such that $M_0 \Rightarrow^* M'$ and $M \leq M'$?

Example 3.2. *Figure 3.3 provides an example of a Petri Net with an initial marking. As usual we represent the places of the Petri Net by circles and the transitions by rectangle whereas the precondition function is represented by arrows from places to transitions and the postcondition function by arrows from transitions to places and the marking is represented by tokens in the places. In that case the marking is the multiset $\langle\langle p1, q1, lock\rangle\rangle$. We have then the following execution from this marking $\langle\langle p1, q1, lock\rangle\rangle \overset{t_1}{\Rightarrow} \langle\langle p2, q1\rangle\rangle \overset{t_2}{\Rightarrow} \langle\langle p1, q1, lock\rangle\rangle \overset{t_3}{\Rightarrow} \langle\langle p1, q2\rangle\rangle$. This Petri Net could represent the execution of two processes that are initially in state $p1$ and $q1$ respectively and that want to access a critical section protected by a lock and when a token is in the place lock it means that the lock is free.*

As mentioned previously there is a strong connection between Petri Nets and VASS. In fact it easy to encode the behavior of a Petri Net with n places in a VASS of dimension n, each of the counter encoding the number of tokens in the places.

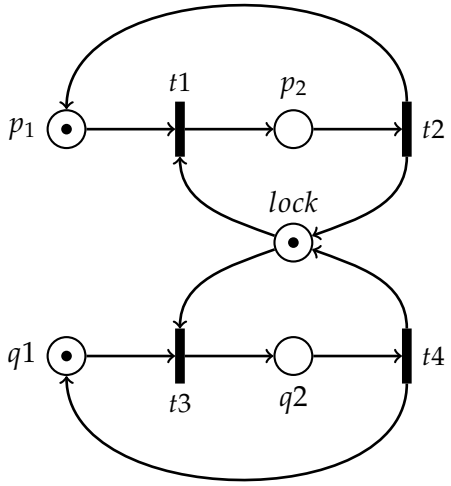

Figure 3.3: An example of Petri Net

Example 3.3. *The VASS represented in Figure 3.4 encodes the behavior of the Petri net of Figure 3.3. Note that when representing graphically VASS we only write the update vectors on the transitions. We see that we have one central state and one state per transition. For each transition of the VASS, there is one transition in charge of performing the action of the precondition and one for the postcondition. Here the places $p1$,$p2$,$lock$,$q1$ and $q2$ are associated to the counters x_1, x_2, x_3, x_4 and x_5 and for instance In the VASS the transitions going in and out of q_i represents the transition t_i of the Petri Net.*

It is as well possible to encode a VASS of dimension n in a Petri net by creating a place for each counter and a place for each control states. In [HP79], a better construction is proposed where one needs only $n + 3$ places to encode the behavior of a VASS. Thanks to these reductions in polynomial time from Petri Nets to VASS and vice versa, we see why these two models are very close and can be considered as equivalent for many decision problems.

We now state some complexity results for the analysis of VASS and Petri Nets. First, note that from the reductions presented above the coverability problem for Petri Nets and the control state reachability problem for VASS can be reduced in polynomial time one into the other. It appears that these two problems are decidable and their complexity is well-known.

Theorem 3.5. *[Lip76; Rac78]* PN-Coverability *and* CS-ControlReach *for VASS are* EXPspace*-complete.*

For what concerns the reachability problem in VASS and Petri Nets, we know that this problem is decidable [May84; Kos82; Lam92; Ler11] but it is a tedious result to obtain. In fact, the algorithms presented in the above mentionned papers are non-elementary and the best know lower bounds for many years was the one for coverability, i.e. EXPspace-hard. However, in [Cze+19b] it has been shown that this problem is effectively non elementary.

Theorem 3.6. *[May84; Kos82; Lam92; Ler11; Cze+19b]* PN-Reach *and* CS-Reach *for VASS are decidable and non-elementary.*

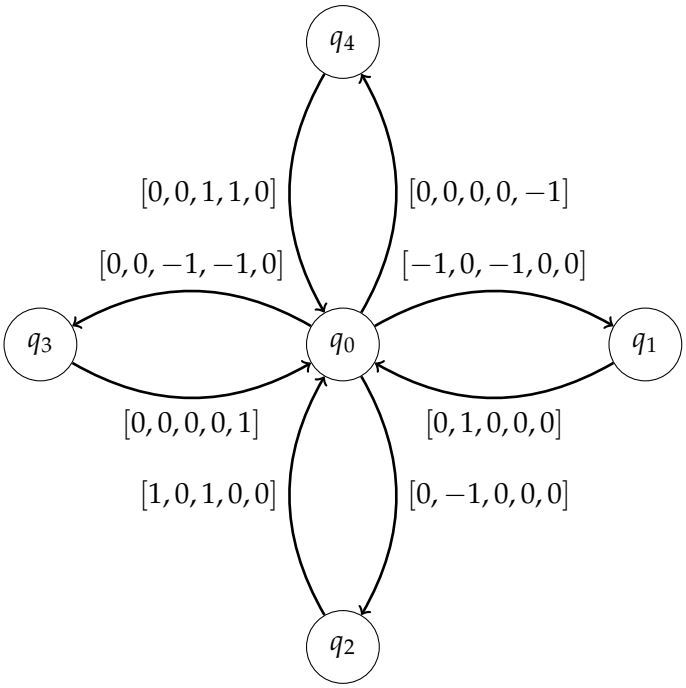

Figure 3.4: A VASS encoding the Petri Net of Figure 3.3

3.3.5 *Flat Counter Systems*

Another way to regain decidability for the verification of counter systems consists in imposing restrictions on the underlying graph structure of the system. An Affine Counter System is *flat* if every node in the underlying graph belongs to at most one simple cycle (a cycle being simple if no edge is repeated twice in it). Hence in a flat Affine Counter System, simple cycles can be organized as a DAG where two simple cycles are in the relation whenever there is path between a node of the first cycle and a node of the second cycle.

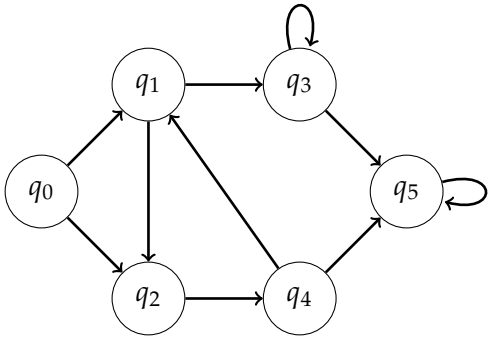

Figure 3.5: Example of a flat system

Example 3.4. *Figure 3.5 provides an example of a flat Affine Counter System where guards and updates have been omitted. We see that each state belongs to at most one simple cycle. For instance, q_1, q_2 and q_4 belong to the same simple cycle.*

In [FO97], the authors show that for flat VASS the reachability relation $\rightarrow^*$ is definable in Presburger arithmetic and the corresponding Presburger formula can effectively be built (it is assumed that the control states are encoded into naturals, and hence each configuration is a vector of naturals). The idea to obtain such result is that each path in the system can be represented by a succession of simple paths (with no repeated states) and loops and the configurations reached by such paths can be expressed by a Presburger formula. Since the system is flat, there is only a finite (but possibly exponential) number of such representations. It is then possible to encode the reachability problems into Presburger arithmetic and to obtain decidability thanks to [Pre29].

Theorem 3.7. *[FO97]* CS-ControlReach *and* CS-Reach *are decidable for flat VASS.*

Note that it is even proven that the reachability relation can be expressed as an existentially quantified formula, but due to the number of paths to enumerate there can be an exponential blowup when building the formula.

it has then been shown that this reasoning can be applied to more general classes of counter systems. For instance in [CJ98] a similar result on the reachability relation is shown for flat counter systems but where the transitions are labelled with relations (instead of functions) between old value and new values of the counters. This result is quite strong but the considered relations can express guards less expressive than in Affine (or Translating) Counter Systems. A simpler proof of this latter result was given in [BIL09]. The result that will interest us the most concerns a class of flat Affine Counter Systems where the updates are restricted. In [FL02], the authors show that for flat Affine Counter Systems with the finite monoid property the reachability relation is definable in Presburger Arithmetic. We present now what it means for an Affince Counter System to have the finite monoid property.

For an Affine Counter System $S = (Q, \mathtt{C}_n, \Delta, \mathbf{l})$ of dimension $n \geq 1$, we consider $\mathcal{M}_S \subseteq \mathbb{Z}^{n\times n}$ to be the smallest set of matrices, closed under product, which contains the identity matrix $\mathbf{I}_n$ and each matrix $\mathbf{A}$ such that $\mathit{update}(\delta) = (\mathbf{A}, \mathbf{b})$ for some $\delta \in \Delta$. Clearly, $\mathcal{M}_S$ forms a monoid with the matrix product as binary operation and $\mathbf{I}_n$ as identity element. We say that S has the *finite monoid property* if the set $\mathcal{M}_S$ is finite. We denote by $\mathcal{FMACS}$ the class of Affine Counter Systems with the finite monoid property. In [FL02], it is proved that the reachability relation $\rightarrow^*$ for flat Affine Counter Systems with the finite monoid property is Presburger definable (here again it is assumed that $Q \subseteq \mathbb{N}$) and that the corresponding formula can be effectively computed. This allows us to deduce the following decidability result.

Theorem 3.8. *[FL02]* CS-ControlReach *and* CS-Reach *are decidable for flat Affine Counter Systems with the finite monoid property.*

Note that if S is a Translating Counter System, then $\mathcal{M}_S = \{\mathbf{I}_n\}$ and this monoid is finite. From the previous theorem, we get automatically the next result.

Corollary 3.1. *[FL02]* CS-ControlReach *and* CS-Reach *are decidable for flat Translating Counter Systems.*

Example 3.5. *Figure 3.6 provides an example of a flat Affine Counter System with the finite monoid property. From the initial state q_0 with all counters equal to 0, this system begins with incrementing $\mathtt{x}_1$ a certain number of times with the transition δ_0 then, with δ_1, it transfers the value of the counter $\mathtt{x}_1$ to $\mathtt{x}_3$ and resets $\mathtt{x}_1$; the loop labelled by δ_2 increments both $\mathtt{x}_1$ and $\mathtt{x}_2$ until they both reach the*

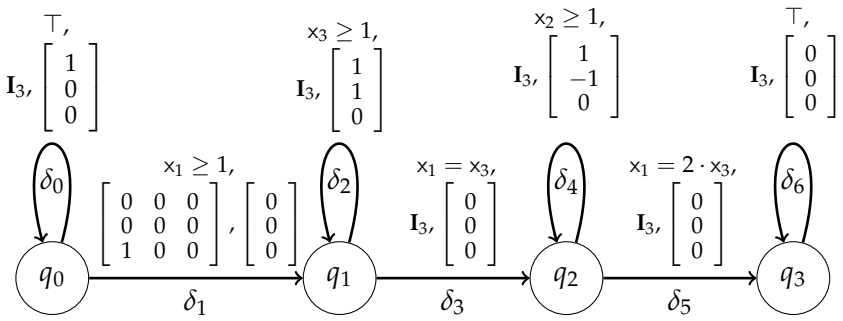

Figure 3.6: A flat Affine Counter System with the finite monoid property

value of x_3 *and finally the loop labelled by* δ_4 *is used to decrement* x_2 *and increment* x_1 *until the value of* x_1 *is twice the value of* x_3*. As a consequence, when the system reaches* x_3 *the value of* x_1 *is twice the value of* x_3 *and the value of* x_2 *is equal to* 0*. Hence, any run reaching* q_3 *visits the state* q_1 *exactly the same number of times as the state* q_2*.*

Remark. *It is true that imposing the structure of the considered counter system to be flat is a strong restriction, however the results presented in [FL02] can be used to compute an under-approximation of the set of executions of an Affine Counter Systems with the finite monoid property by enumerating its flat unfolding. The method consists in deleting transitions or unfolding some loops a finite number of times to obtain a flat subsystem of the Counter System under analysis and then compute its reachability transition.*

3.4 SUMMARY OF THE CLASSES OF SYSTEMS

The table 3.1 recapitulate the different classes of counter systems we have presented in this chapter.

Acronym	Class	Guards	Updates
$\mathcal{ACS}$	Affine Counter Systems	Full guards	Affine functions
$\mathcal{KS}$	Kripke Structures	No Guard	No Update
$\mathcal{TCS}$	Translating Counter Systems	Full Guards	Translations
$\mathcal{OCS}$	One Counter Systems	$x_1 \sim b$	Translations
simple $\mathcal{OCS}$	Simple One Counter Systems	$true, x_1 = 0$	Translations
$\mathcal{VASS}$	Vector Addition Systems with States	$true$	Translations
$\mathcal{FMACS}$	Affine Counter Systems with the finite monoid property	Full guards	Affine functions whose associated monoid is finite

Table 3.1: Summary of the classes of counter systems

4

VERIFICATION OF LINEAR TIME PROPERTIES

In this chapter, we present the results we have obtained concerning the model-checking of counter systems with linear time properties. As we have seen in the previous chapter, even if reachability is undecidable for simple classes of counter systems, decidability can be regained by imposing some restrictions on the considered systems and many research works have been done to obtain a precise complexity characterization of the complexity of the reachability problems under these restrictions.

In order to specify more complex properties, one way consists in relying on specification languages which allow to describe the evolution of the system during the time. Most of the time such specification languages allow to express reachability properties and much more, hence it makes sense to study the model-checking problem for complex specification languages for the cases where at least the control state reachability problem is known to be decidable.

The works we present here focus on linear time specifications which describe behaviors of runs of systems. The model-checking problem asks then given a system and a specification whether the runs of the system satisfy the specification. Previous works have studied such a model-checking problem over different classes of counter systems. For instance, in [Hab97] it is shown that the model-checking of linear μ-calculus and of the Linear time Temporal Logic (LTL) over VASS is EXPSPACE-complete. The model-checking problem of LTL over Simple One Counter Systems has been studied in [Göl+10] and it is proved that it is PSPACE-complete. It means that for this latter model the complexity is the same as for Kripke structures [SC85].

We looked at the model-checking problem of linear time properties over the class of flat counter systems. This research direction was motivated by two facts. First, in [KF11] it is shown that the model-checking of LTL over flat Kripke structures (the paper mentions weak structures, which is an equivalent term for flat) is in NP, hence in this case considering flat structures simplifies the model-checking algorithm. Second in [Dem+10] it is shown that the model-checking problem for an extension of the branching time temporal logic CTL$*$ where the atomic propositions are Presburger definable sets of configurations is decidable for flat Affine Counter Systems with the finite monoid property. Actually, they prove a more general result by considering flat counter systems whose updates are functions and their counting iteration over cycles in the structures is Presburger definable. However no precise complexity bound is given and for instance for flat Translating Counting Systems, their procedure leads to a very rough upper bound in 4EXPTIME. In fact, the decision algorithm translates the model-checking problem into a Presburger formula of exponential size. Hence from these results, we decided to investigate the complexity of the model-checking problem for flat counter systems in order to get precise bounds and eventually decision procedures of practical use.

CONTRIBUTIONS. Our first works [DDS12; DDS15] on this subject considered the model-checking of the linear time temporal logic LTL with past operators (Past LTL) over flat

Translating Counter Systems. We considered an extension of Past LTL which allows as well to characterize the counter values at some positions in the runs thanks to arithmetical constraints used as atomic propositions. We showed that the model-checking problem of this logic over Translating Counter System is NP-complete (hence it as a complexity very close to the case of flat Kripke structures).

Afterwards in [DDS13], we have considered other formalisms specifying linear-time properties (first order logic, linear μ-calculus, infinite automata) and our goal was to determine the complexity of model-checking problems over flat Translating Counter Systems. Note that first order logic is as expressive as Past LTL but much more concise whereas linear μ-calculus is strictly more expressive than Past LTL, which motivates the choice for these formalisms dealing with linear properties. Here again our formalisms admit arithmetical constraints about counter values in the specification. We obtained the following results for the model-checking over flat Transalting Counter Systems: the problem is PSPACE-complete for first order logic (this is very surprising since the model-checking of classical first-order formulas over arbitrary Kripke structures is known to be non-elementary), the problem is PSPACE-complete for linear μ-calculus and the problem is NP-complete if the specification is given by a Büchi automaton. Not only we obtained tight complexity bounds but we proposed a general framework to deal with the different specification languages that could possibly be reused. Note that thanks to this framework we could as well reprove the results obtained for Past LTL.

Finally, in [IS16], we looked in which measure the previously developed techniques could be adapted to the case of flat Affine Counter Systems with the finite monoid property. Most of the techniques for the model-checking problem could be reused from the previous works however the difficulty here was to understand better how to characterize the reachability relations in an efficient way in order to obtain a good complexity bound. As a matter of fact, we first provided a complexity bound for the reachability problem in such systems. Furthermore, in order to avoid complexity in the reasonning due to the guards in the system, we had to assume that the guards were only conjunctions of linear constraints. We showed that for such systems the reachability problem and the model-checking of Past LTL both belong to the second level of the polynomial hierarchy Σ_2^p whereas the model-checking of first order logic formulas is PSPACE-complete.

4.1 SPECIFICATION LANGUAGES AND MODEL-CHECKING PROBLEM

We introduce first a general definition of linear time specification language dedicated to counter systems and then we present the different languages we will consider.

As for counter systems, we consider $\mathtt{C} = \{\mathtt{x}_1, \mathtt{x}_2, \ldots\}$ an infinite set of counters (variables interpreted over non-negative integers), $\mathtt{C}_n = \{\mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{x}_n\}$ its restriction to n counters and $\mathrm{AT} = \{\mathtt{p}_1, \mathtt{p}_2, \ldots\}$ a countably infinite set of propositional variables. Since our specifications will allow to reason both on the atomic propositions labelling the states of the counter systems under study and the counter values, we propose a general way to treat these two notions uniformly. For this matter, we use constrained alphabets whose letters should be understood as Boolean combinations of atomic formulas (details follow). A *n-dim constrained alphabet* (shortly constrained alphabet) is a triple of the form (at, ag_n, Γ) where:

- at is a finite subset of AT,

- ag_n is a finite subset of atomic guards from $\mathtt{G}(\mathtt{C}_n)$, i.e. guards of the form $\sum_{i=1}^{n} a_i \cdot \mathtt{x}_i \sim b$ where the a_i's are in $\mathbb{Z}$ and b in $\mathbb{N}$,
- Γ is a subset of $2^{at \cup ag_n}$.

The size of a constrained alphabet is given by $\mathtt{size}(at, ag_n, \Gamma) = card(at) + \Sigma_{g \in ag_n} \mathtt{size}(g) + card(\Gamma)$. An important feature to evaluate the size of (at, ag_n, Γ) is to notice that at, ag_n and Γ are finite sets and Γ is not necessarily equal to the power set $2^{at \cup ag_n}$. Our specification languages over such a constrained alphabet will then recognize infinite words in Γ^*.

A *specification language* $\mathcal{L}$ over a constrained alphabet (at, ag_n, Γ) is a set of *specifications* A, each of it defining a set $L(A)$ of infinite words in Γ^ω. Hence the language associated to a specification is a set of infinite words where at each position we find a subset of atomic propositions and of atomic guards. Note that we explicitly introduce the notions of specification languages and specifications for the sake of clarity in forthcoming definitions, but they cover what is commonly accepted. We will also sometimes consider specification languages over (unconstrained) standard finite alphabets (as usually defined).

It remains to explain how the runs of an Affine Counter System and a specification are related. We consider an Affine Counter System $S = (Q, \mathtt{C}_n, \Delta, \mathbf{l})$ and a constrained alphabet (at, ag_n, Γ). Given an infinite run $\rho := (q_0, \mathbf{v}_0) \to (q_1, \mathbf{v}_1) \cdots$ of S (with $q_i \in Q$ and $\mathbf{v}_i \in \mathbb{N}^n$ for all $i \in \mathbb{N}$) and an infinite word $w = a_0 a_1 \ldots \in \Gamma^\omega$, we say that ρ *satisfies* w, written $\rho \models w$ whenever for every $i \geq 0$, we have:

1. $\mathtt{p} \in \mathbf{l}(q_i)$ for every $\mathtt{p} \in (a_i \cap at)$,
2. $\mathtt{p} \notin \mathbf{l}(q_i)$ for every $\mathtt{p} \in (at \setminus a_i)$,
3. $\mathbf{v}_i \models \mathtt{g}$ for every $\mathtt{g} \in (a_i \cap ag_n)$,
4. $\mathbf{v}_i \not\models \mathtt{g}$ for every $\mathtt{g} \in (ag_n \setminus a_i)$.

Intuitively at each position i of the run, we have that an atomic proposition in at holds if and only if it is present in a_i and that a guard in ag_n is satisfied by the counter values if and only if it is present in a_i. For a specification A such that $L(A) \subseteq \Gamma^\omega$, we say that a run ρ satisfies A, denoted by $\rho \models A$, if and only if there exists a word $w \in L(A)$ such $\rho \models w$.

We have now the tools to define the model-checking problem for linear-time properties over Affine Counter System which is at the heart of this chapter. This problem is parameterised by a specification language $\mathcal{L}$ over a constrained alphabet (at, ag_n, Γ).

CS-MODEL-CHECKING($\mathcal{L}$)

Input: An Affine Counter System $S = (Q, \mathtt{C}_n, \Delta, \mathbf{l})$,
an initial configuration $c_0 \in Q \times \mathbb{N}^n$,
and a specification A from $\mathcal{L}$;

Question: Does there exists an infinite run ρ starting at c_0 such that $\rho \models A$?

Note that the model-checking problem we study corresponds to the existential version of the model-checking, i.e. we seek for a run satisfying a specification, whereas usually it is more natural to consider the universal version where one wants to verify that all the runs satisfy a specification. The reason for this choice is that from an algorithmic point of view it fits better to our framework. Furthermore, most specification languages we study

are closed under complement hence our result can be adapted to deal with the universal model-checking problem. However is some cases, complementing the specification to go from the universal model-checking problem to the existential one might have a cost and it could affect the complexity of the procedure.

4.2 A BUNCH OF SPECIFICATION LANGUAGES

We now present the different specification languages for which we obtain results for the model-checking of flat counter systems. They can be separated in two categories, in the first one we use automata to describe the specification, whereas in the second one we use logical formalisms.

4.2.1 *Automata based specifications*

We present here two specification languages BS (Büchi Specifications) and ABS (Alternating Büchi Specifications). To define these languages, we rely on (nondeterministic) Büchi automata and alternating Büchi automata. However, even if the specification languages are very close to the automata mechanism they rely on, we shall distinguish between the specifications and their underlying automata. Moreover, nondeterministic Büchi automata and alternating Büchi automata are known to have the same expressive power but not the same conciseness and therefore it makes sense to distinguish them.

We first recall the definition of Büchi Automata.

Definition 4.1 (Büchi Automaton)**.** *A Büchi automaton B is a tuple $(Q, \Sigma, \delta, q_0, F)$ where:*

- *Q is a non-empty finite set of states,*
- *Σ is a finite alphabet,*
- *$\delta \subseteq Q \times \Sigma \times Q$ is the finite set of transitions,*
- *$q_0 \in Q$ is the initial state, and,*
- *$F \subseteq Q$ is the set of final states.*

An accepting run ρ in B for a word $w \in \Sigma^\omega$ is as an infinite sequence of states $\rho \in Q^\omega$ such that $(\rho(i), w(i+1), \rho(i+1)) \in \delta$ for every $i \in \mathbb{N}$ and $\rho(j) \in F$ for infinitely many j's. We define then the language of B, denoted by $L(B)$ as the set of infinite words $\{w \in \Sigma^\omega \mid$ there exists an accepting run in B for $w\}$.

Let us fix a constrained alphabet (at, ag_n, Γ) In order to be a bit concise, we define a specification languages where we allow the transitions to be labelled by Boolean combinations of atoms from $at \cup ag_n$, leading to the specifications in BS (Büchi Specifications). First we define $\mathbb{B}(at \cup ag_n)$ the set of formulas obtained by Boolean combinations over $at \cup ag_n$ thanks to the following grammar:

$$\psi := at \mid ag_n \mid \neg\psi \mid \psi \wedge \psi$$

Then we say that a specification A is in BS if it is of the form $(Q, E.q_0, F)$ where Q is a finite set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of accepting states and E is a finite subset of $Q \times \mathbb{B}(at \cup ag_n) \times Q$. A specification A in BS is just a concise representation for the Büchi automaton $B_A = (Q, 2^{at \cup ag_n}, \delta, q_0, F)$ where δ is a subset of $Q \times 2^{at \cup ag_n} \times Q$ and $(q, a, q') \in \delta$ if and only if there is $(q, \psi, q') \in E$ such that $a \models \psi$ in the propositional

sense (an atom in $(at \cup ag_n) \setminus a$ is interpreted by false). We say that A is over the constrained alphabet (at, ag_n, Γ), whenever for all edges $(q, \psi, q') \in E$, we have that ψ holds at most for valuations/letters from Γ. Finally, the language $L(A) \subseteq \Gamma^\omega$ is defined as $L(B_A)$. Strictly speaking, specifications in BS and Büchi automata over the alphabet Γ are not identical objects but the first ones can be seen as a concise representations of the second ones (an edge in A can lead to an exponential number of transitions in B_A).

We now provide the definition of the specification language based on alternating Büchi automata. Given a finite set Q, we write $\mathbb{B}^+(Q)$ to denote the set of positive Boolean formulas built over Q which respect the following grammar:

$$\psi \ ::= \ \bot \ | \ \top \ | \ q \ | \ \psi \vee \psi \ | \ \psi \wedge \psi$$

where $q \in Q$. Every subset $Y \subseteq Q$ can be viewed as a propositional valuation such that $q \in Y$ if and only if q is interpreted as true. We write $Y \models \psi$ with $A \in \mathbb{B}^+(Q)$ to denote that ψ holds true under the propositional valuation induced by the subset Y.

Definition 4.2 (Alternating Büchi Automaton)**.** *An alternating Büchi automaton B is a tuple $(Q, \Sigma, \delta, q_0, F)$ where:*

- *Q is a non-empty finite set of states,*
- *Σ is a finite alphabet,*
- *$\delta : Q \times \Sigma \mapsto \mathbb{B}^+(Q)$ is the transition function,*
- *$q_0 \in Q$ is the initial state, and,*
- *$F \subseteq Q$ is the set of accepting states.*

A run ρ for the ω-word $a_0 a_1 a_2 \ldots \in \Sigma^\omega$ is a (possibly infinite) directed acyclic graph (V, R) where $V \subseteq Q \times \mathbb{N}$ is the set of vertices and $R \subseteq V \times V$ is the set of edges which satisfies the following properties:

- $(q_0, 0) \in V$,
- $R \subseteq \bigcup_{l \geq 0} (Q \times \{l\}) \times (Q \times \{l+1\})$,
- For every $(q, l) \in V \setminus \{(q_0, 0)\}$, there is $q' \in Q$ such that $((q', l-1)(q, l)) \in R$.
- For every $(q, l) \in V$ we have $\{q' \mid ((q, l), (q', l+1)) \in R\} \models \delta(q, a_l)$.

A run ρ is accepting whenever every infinite path through that run defines an ω-word in Q^ω such that a state from F is visited infinitely often. Similarly to BS, a specification A in ABS over the constrained alphabet (at, ag_n, Γ) is of the form (Q, E, q_0, F) where Q is a finite set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of accepting states and E is the partial transition function $Q \times \mathbb{B}(at \cup ag_n) \to \mathbb{B}^+(Q)$ and for each $\phi \in \mathbb{B}(at \cup ag_n)$, we have that ϕ holds at most for valuations/letters from Γ. Following the same reasoning as for BS, a specification A in ABS is only a concise representation for an alternating Büchi automaton $B_A = (Q, \Gamma, \delta, q_0, F)$ where $\delta(q, a) = E(q, \phi)$ whenever $a \models \phi$ in the propositional sense (an atom in $(at \cup ag_n) \setminus a$ is interpreted by false). The language $L(A)$ is then exactly equal to $L(B_A)$.

Note that these two specification languages have the same expressive power however the ABS are more concise than BS.

4.2.2 *Linear Time Logics*

We now present three logical languages that are tailored to specify properties of runs of counter systems: Past LTL (see e.g. [SC85]), first order logic and the linear μ-calculus, see e.g. [Var88]. A specification in one of these logical specification languages is just a formula. The main differences with their standard versions in which models are ω-sequences of propositional valuations is that here atomic formulas are either propositional variables in AT or atomic guards in $\mathtt{G}(\mathtt{C}_n)$.

4.2.2.1 *Past LTL*

We now provide the syntax and semantics of Past LTL. Formulas of Past LTL over the constrained alphabet (at, ag_n, Γ) are defined by the following grammar:

$$\phi ::= \mathtt{p} \mid \mathtt{g} \mid \neg\phi \mid \phi \wedge \phi \mid \mathtt{X}\phi \mid \mathtt{X}^{-1}\phi \mid \phi \mathtt{U} \phi \mid \phi \mathtt{S} \phi$$

where $\mathtt{p}$ belongs to at and $\mathtt{g}$ belongs to ag_n. A formula ϕ of Past LTL is interpreted over an ω-word w in Γ^ω at a certain position $i \in \mathbb{N}$ and the temporal operator $\mathtt{X}$, $\mathtt{X}^{-1}$, $\mathtt{U}$ and $\mathtt{S}$ allows to navigate in this word. $\mathtt{X}$ is used to move to the next position, $\mathtt{X}^{-1}$ to the previous position, $\mathtt{U}$ is used to move to a position in the 'future' ensuring that some properties hold along the path and $\mathtt{S}$ has a similar role but to move to a position in the past. This is formalized by the satisfaction relation $\models$ which is defined inductively as follows:

$$\begin{aligned}
w, i \models \mathtt{p} &\overset{\text{def}}{\Leftrightarrow} \mathtt{p} \in w(i) \\
w, i \models \mathtt{g} &\overset{\text{def}}{\Leftrightarrow} \mathtt{g} \in w(i) \\
w, i \models \neg\phi &\overset{\text{def}}{\Leftrightarrow} w, i \not\models \phi \\
w, i \models \phi_1 \wedge \phi_2 &\overset{\text{def}}{\Leftrightarrow} w, i \models \phi_1 \text{ and } w, i \models \phi_2 \\
w, i \models \mathtt{X}\phi &\overset{\text{def}}{\Leftrightarrow} w, i+1 \models \phi \\
w, i \models \phi_1 \mathtt{U} \phi_2 &\overset{\text{def}}{\Leftrightarrow} w, j \models \phi_2 \text{ for some } i \leq j \\
&\qquad \text{such that } w, k \models \phi_1 \text{ for all } i \leq k < j \\
w, i \models \mathtt{X}^{-1}\phi &\overset{\text{def}}{\Leftrightarrow} i > 0 \text{ and } w, i-1 \models \phi \\
w, i \models \phi_1 \mathtt{S} \phi_2 &\overset{\text{def}}{\Leftrightarrow} w, j \models \phi_2 \text{ for some } 0 \leq j \leq i \\
&\qquad \text{such that } w, k \models \phi_1 \text{ for all } j < k \leq i
\end{aligned}$$

Given a Past LTL formula ϕ, we define $L(\phi)$ as $\{w \in \Gamma^\omega \mid w, 0 \models \phi\}$. Finally, the size of a Past LTL formula is understood as the number of subformulas.

4.2.2.2 *First Order logic (FO)*

The second logical formalism we consider are first order logic formulas where the variables are interpreted over the positions of an ω-word and which use atomic propositions or atomic constraints as unary predicates and the successor relation as binary predicates. To simplify we denote FO this logic in our context. Using Kamp's Theorem [Kam68], first-order logic has the same expressive power as Past LTL, however it is more concise, in fact it is known that the satisfiability problem for first-order logic formulas is non-elementary and consequently the translation into Past LTL leads to a significant blow-up in the size of the formula.

Formulas of FO over the constrained alphabet (at, ag_n, Γ) are defined by the grammar below:

$$\phi \quad ::= \quad \mathtt{p}(\mathtt{z}) \;\mid\; \mathtt{g}(\mathtt{z}) \;\mid\; \mathtt{succ}(\mathtt{z},\mathtt{z}') \;\mid\; \mathtt{z} < \mathtt{z}' \;\mid\; \mathtt{z} = \mathtt{z}' \;\mid\; \neg\phi \;\mid\; \phi \wedge \phi' \;\mid\; \exists \mathtt{z}\, \phi(\mathtt{z})$$

where $\mathtt{z}$ is a propositional variable from $\mathtt{VAR}$ such that $\mathtt{VAR}$ is a countably infinite set of variables that is disjoint from AT and $\mathtt{p} \in at$ and $\mathtt{g} \in ag_n$. A variable $\mathtt{z}$ is free in a FO formula ϕ if and only if $\mathtt{z}$ does not occur within the scope of any quantifier ($\exists$) in ϕ. As usual, we write $\mathtt{free}(\phi)$ to denote the set of free variables in the formula ϕ. A formula with no free variable is called a sentence. The quantifier height $qh(\phi)$ of a formula ϕ is then the maximum nesting depth of the operators $\exists$ in ϕ.

As already mentioned, the variables in an FO formula represent positions in the word, and the binary predicates $<$ and $=$ is the classical order relation, respectively the equality on positions, whereas $\mathtt{succ}$ is used to say that a position is the next position of another one. Models of FO formulas are then again ω-words in Γ^ω. We define position assignment as a partial function $f : \mathtt{VAR} \to \mathbb{N}$. Given a model $w \in \Gamma^\omega$, a FO formula ϕ and a position assignment f such that $f(\mathtt{z}) \in \mathbb{N}$ for every variable $\mathtt{z} \in free(\phi)$, the satisfaction relation $\models_f$ is defined as follows:

$$\begin{array}{lll}
w \models_f \mathtt{p}(\mathtt{z}) & \text{iff} & \mathtt{p} \in w(f(\mathtt{z})) \\
w \models_f \mathtt{g}(\mathtt{z}) & \text{iff} & \mathtt{g} \in w(f(\mathtt{z})) \\
w \models_f \mathtt{succ}(\mathtt{z},\mathtt{z}') & \text{iff} & f(\mathtt{z}') = f(\mathtt{z}) + 1 \\
w \models_f \mathtt{z} < \mathtt{z}' & \text{iff} & f(\mathtt{z}) < f(\mathtt{z}') \\
w \models_f \mathtt{z} = \mathtt{z}' & \text{iff} & f(\mathtt{z}) = f(\mathtt{z}') \\
w \models_f \neg\phi & \text{iff} & w \not\models_f \phi \\
w \models_f \phi \wedge \phi' & \text{iff} & w \models_f \phi \text{ and } w \models_f \phi' \\
w \models_f \exists \mathtt{z}\, \phi(\mathtt{z}) & \text{iff} & \text{there exists } j \in \mathbb{N} \text{ such that } w \models_{f[\mathtt{z} \to j]} \phi(\mathtt{z})
\end{array}$$

where $f[\mathtt{z} \to j]$ is the position assignment g such that $g(\mathtt{z}') = f(\mathtt{z}')$ for all $z' \in dom(f) \setminus \{\mathtt{z}\}$ and $g(\mathtt{z}) = j$. As for Past LTL , for a FO formula ϕ, we define $L(\phi)$ as $\{w \in \Gamma^\omega \mid w \models \phi\}$ and the size of a formula is understood as the number of its subformulas.

4.2.2.3 *Linear μ-calculus*

The last logical formalism we present is the linear μ-calculus. It is the most expressive logic for which we study the model-checking problem and it has the same expressive power as Büchi automata whereas Past LTL and FO are less expressive (see for instance [Wol83; Var88]). Even if it is very expressive, it suffers from an important drawback: it is difficult to have an intuition on the signification of formulas when many nested fixpoints are used. Formulas of the linear μ-calculus over the constrained alphabet (at, ag_n, Γ) are defined by the grammar below:

$$\phi ::= \mathtt{z} \;\mid\; \mathtt{p} \;\mid\; \mathtt{g} \;\mid\; \neg\phi \;\mid\; \phi \wedge \phi \;\mid\; \phi \vee \phi \;\mid\; \mathtt{X}\phi \;\mid\; \mathtt{X}^{-1}\phi \;\mid\; \mu\mathtt{z} \cdot \phi$$

where $\mathtt{z}$ is a propositional variable from $\mathtt{VAR}$ such that $\mathtt{VAR}$ is a countably infinite set of variables that is disjoint from AT and $\mathtt{p} \in at$ and $\mathtt{g} \in ag_n$. Only well-formed formulas are really considered and taken care of. A formula ϕ is well-formed if and only if for all $\mu\mathtt{z} \cdot \psi$ in ϕ, the following conditions are verified:

1. for all subformulas $\mu z \cdot \psi'$, we have $\psi = \psi'$,
2. z is not free in ϕ and
3. every occurrence of z in the syntax tree of ψ is under an even number of negation symbols in ψ.

Note that, given any linear μ-calculus formula, we can easily transform it to satisfy the first two conditions. We include these conditions in the definition of "well-formed" formulas to be precise about the form of the formula. In the rest of the document we only consider well-formed linear μ-calculus formulas. As usual $\nu z \cdot \phi$ is defined as $\neg \mu z \cdot \neg \phi[z \leftarrow \neg z]$ (where $\phi[z \leftarrow \neg z]$ indicates that we substitute all the occurrences of z in ϕ with $\neg z$.

A formula ϕ of linear μ-calculus is interpreted over an ω-word w in Γ^ω at a certain position $i \in \mathbb{N}$ and the satisfaction relation $\models$ is parameterised by maps of the form $f : \mathtt{VAR} \mapsto 2^{\mathbb{N}}$ assigning set of positions to a variable. We have then:

$$
\begin{array}{lcl}
w, i \models_f \mathtt{z} & \stackrel{\text{def}}{\Leftrightarrow} & i \in f(\mathtt{z}) \\
w, i \models_f \mathtt{p} & \stackrel{\text{def}}{\Leftrightarrow} & \mathtt{p} \in w(i) \\
w, i \models_f \mathtt{g} & \stackrel{\text{def}}{\Leftrightarrow} & \mathtt{g} \in w(i) \\
w, i \models_f \neg \phi & \stackrel{\text{def}}{\Leftrightarrow} & w, i \not\models_f \phi \\
w, i \models_f \phi_1 \wedge \phi_2 & \stackrel{\text{def}}{\Leftrightarrow} & w, i \models_f \phi_1 \text{ and } w, i \models_f \phi_2 \\
\rho, i \models_f \mathtt{X}\phi & \stackrel{\text{def}}{\Leftrightarrow} & w, i+1 \models_f \phi \\
\rho, i \models_f \mathtt{X}^{-1}\phi & \stackrel{\text{def}}{\Leftrightarrow} & i > 0 \text{ and } w, i-1 \models_f \phi
\end{array}
$$

It remains to define the relation for formulas whose outermost connective is the fixpoint operator. Fixpoints in linear μ-calculus are considered as monotone functions over the complete lattice $(2^{\mathbb{N}}, \subseteq)$. Given an ω-word in Γ^ω and a linear μ-calculus formula ϕ, we first define the least fixpoint of the monotone function $\mathcal{F}_{f,w} : 2^{\mathbb{N}} \mapsto 2^{\mathbb{N}}$ by $\mathcal{F}_{f,w}(Y) = \{i \in \mathbb{N} \mid w, i \models_{f[z \to Y]} \phi\}$. This can be computed iteratively as a sequence $(Z_j)_{j \in \mathbb{N}}$ of sets of positions verifying:

- $Z_0 = \emptyset$ and
- $Z_{j+1} = \mathcal{F}_{f,w}(Z_j)$ for all $j \geq 0$.

and define the least fixpoint as $Z = \bigcup_{j \in \mathbb{N}} Z_j$. It is well-known, by Knaster-Tarski's Theorem [Kna28; Tar55], that the least fixed point Z exists. So, the satisfaction relation is defined as follows:

$$w, i \models_f \mu z \cdot \phi \quad \stackrel{\text{def}}{\Leftrightarrow} \quad i \in Z \text{ where } Z \text{ is the least fixpoint of the monotone function } \mathcal{F}_{f,w}$$

Note that νz can be defined similarly but using the greatest fixpoint instead of the least fixpoint. As for Past LTL and FO, for a well-formed linear μ-calculus formula ϕ, we define $L(\phi)$ as $\{w \in \Gamma^\omega \mid w, 0 \models \phi\}$ and the size of a formula is understood as the number of its subformulas.

4.3 EXAMPLES

In Figure 4.1, we present a simple flat Translating Counter System of dimension 2. We assume that the set of atomic propositions AT is equal to $\{q_0, q_1, q_2, q_3, q_4, q_5\}$ and the

labelling function $\mathbf{l}$ is such that $\mathbf{l}(q_i) = \{q_i\}$ for all $i \in [0,5]$. Furthermore, when we do not write anything above an edge δ it means that the associated guard $guard(\delta)$ is equal to $\top$ (true) and the update vector is $update(\delta) = [0,0]$. For this counter system, a specification in Past LTL could be $\phi = \top \,\mathtt{U}\, (q_4 \wedge \mathtt{X} q_3 \wedge x_1 > 5)$ which states that there exists a run where the control state q_4 is visited with the value for the first counter strictly bigger than 5 and the next state in this run is q_3. An example of run ρ starting at the configuration $(q_0, [0,0])$ satisfying ϕ is for instance:

$$\rho := (q_0,[0,0]) \to (q_0,[1,0]) \to (q_2,[2,0]) \to (q_3,[3,5]) \to (q_4,[5,9]) \to$$
$$(q_3,[5,9]) \to (q_4,[7,13]) \to (q_3,[7,13]) \to (q_5,[10,14]) \to (q_5,[10,14]) \cdots$$

This specification could as the well be expressed in FO thanks to the following formula $\phi' = \exists z_1 . \exists z_2 . q_4(z_1) \wedge q_3(z_2) \wedge \mathtt{succ}(z_1, z_2) \wedge (x_1 > 5)(z_1)$

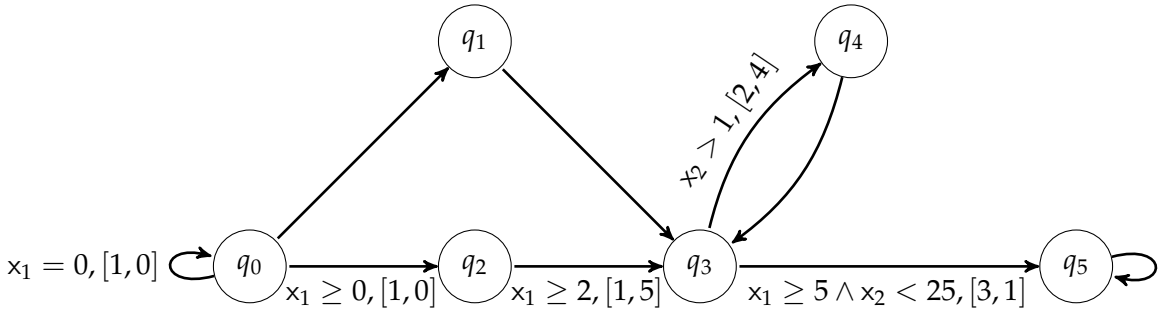

Figure 4.1: A flat Translating Counter System

In Figure 4.2, we give another simple Translating Counter System with two counters and with labeling function $\mathbf{l}$ such that $\mathbf{l}(q_3) = \{\mathtt{p}, \mathtt{q}\}$ and $\mathbf{l}(q_5) = \{\mathtt{p}\}$. We would like to check for some configuration $c_0 = (q_0, \mathbf{v})$ with $\mathbf{v} \in \mathbb{N}^2$ whether there is some infinite run from c_0 for which after some position i, all future even positions j (i.e. $i = j \mod 2$) satisfy that $\mathtt{p}$ holds and the first counter is equal to the second counter.

The specification in BS for this property is presented in Figure 4.3 (as it is usually done, we indicate by a double circle the accepting states). This property can as well be specified in linear μ-calculus using as atomic formulas either propositional variables or atomic guards. The corresponding formula in linear μ-calculus is: $\mu z_1 . (\nu z_2 . (\mathtt{p} \wedge (x_1 - x_2 = 0) \wedge \mathtt{X}\mathtt{X} z_2) \vee \mathtt{X} z_1)$.

Clearly, a position verifying such a property occurs in any run after reaching the control state q_2 with the same value for both counters. Hence, the configurations $(q_0, \mathbf{v})$ from which there exists an infinite run satisfying the above mentioned property have counter values $\mathbf{v} \in \mathbb{N}^2$ verifying the Presburger formula below:

$$\exists y . (((x_1 = 3y + x_2) \wedge (\forall y' . g(x_2 + y', x_2 + y') \wedge g'(x_2 + y', x_2 + y' + 1))) \vee$$
$$((x_2 = 2y + x_1) \wedge (\forall y' . g(x_1 + y', x_1 + y') \wedge g'(x_1 + y', x_1 + y' + 1))))$$

The disjunction is designed so that we take into account whether the run visits q_1 or q_3. The quantification over the variable y' corresponds to the infinite number of visits of the loop passing via q_2 and q_4.

In our works, we design a global method which allows to compute systematically for flat Translating Counter System such formulas (even without universal quantifications) for linear time specification languages.

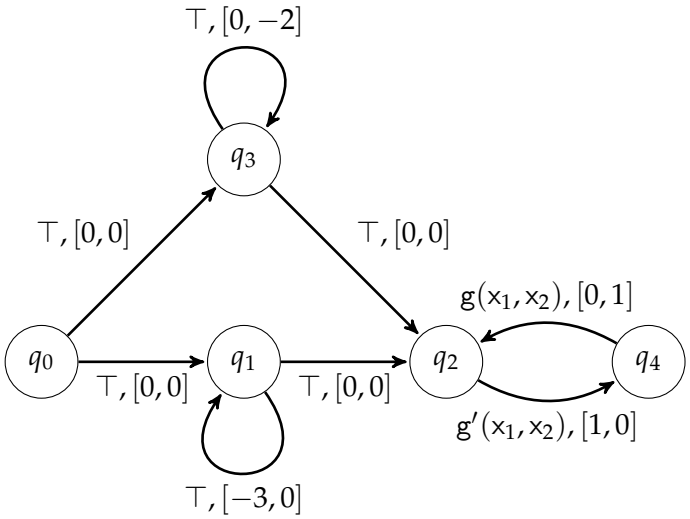

Figure 4.2: A flat Translating Counter System

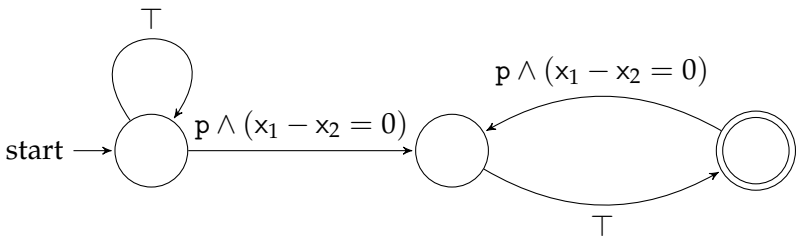

Figure 4.3: A specification in BS

4.4 A GENERAL MODEL-CHECKING ALGORITHM FOR FLAT TRANSLATING COUNTER SYSTEMS

In this section, we present the method we have developed in [DDS13] to perform the model-checking of linear time properties over flat Translating Counter Systems. This method uses the following formal tools. First, since the considered systems are flat, we are able to propose a finite set representations of all the runs using structures we call constrained path schemas. These structures have a specific shape, namely they correspond to a succession of finite paths and loops and we associate to them a quantifier free Presburger formula specifying the number of times each loop can be taken to form a run. We observe then that for the linear time specifications we study,it is possible to bound the number of times each loop is taken, this method can be seen as an adaptation of the stuttering method proposed for LTL in [KS05]. The last formal tool we rely on uses the fact that for some equations systems over the naturals, one can obtain small solutions [BT76]. The combination of these three tools provides us with a general framework to perform model-checking.

4.4.1 *Constrained Path Schemas: a structure to represent runs*

In [DDS15] we introduce minimal path schemas for flat Translating Counter Systems. A path schema $\mathtt{P}$ is an ω-regular expression over the alphabet of transitions of the form $p_1(l_1)^* \cdots p_{k-1}(l_{k-1})^* p_k(l_k)^\omega$ where each p_i is a finite path and each l_i is a simple cycle of the given flat counter system. Such path schemas have been used as well in [LS04; LS05] where they were called linear path schemes. A path schema $\mathtt{P}$ is said to be *minimal* when no transition of the counter system is used more than twice in the expression. Minimal path schemas enjoy the following properties: for a given flat Translating Counter System there are a finite (exponential) number of minimal path schemas and the word of transitions of each run of a flat Translating Counter System is accepted by a minimal path schema. Furthermore, we can associate to each path schema an arithmetical constraint characterizing how many times each loop is taken and thus they represent in a finite manner all the possible runs of a flat counter system.

In [DDS13] we introduce *constrained path schemas* that are more abstract than path schemas. A *constrained path schema* $\mathtt{cps}$ over a constrained alphabet (at, ag_n, Γ) is a pair :

$$(p_1(l_1)^+ \cdots p_{k-1}(l_{k-1})^+ p_k(l_k)^\omega, \phi(\mathtt{y}_1, \ldots, \mathtt{y}_{k-1}))$$

where the first component is an ω-regular expression with $p_i \in \Gamma^*$, $l_i \in \Gamma^+$, and $\phi(\mathtt{y}_1, \ldots, \mathtt{y}_{k-1}) \in \mathtt{G}(\{\mathtt{y}_1, \ldots, \mathtt{y}_{k-1}\})$ is a constraint specifying the number of times each loop l_i in the expression can be taken. We recall that $\mathtt{G}(\{\mathtt{y}_1, \ldots, \mathtt{y}_{k-1}\})$, is made of Boolean combinations of atomic guards of the form $\sum_{i=1}^{k-1} a_i \cdot \mathtt{y}_i \sim b$ where the a_i's are in $\mathbb{Z}$, $b \in \mathbb{N}$ and $\sim \in \{=, \leq, \geq, <, >\}$. Such a constrained path schema defines a language $L(\mathtt{cps}) \subseteq \Gamma^\omega$, that is a subset of the language defined by the ω-regular expression by taking into account the constraints on the repetition of the l_i's. More specifically, this language is defined as follows:

$$L(\mathtt{cps}) = \{p_1(l_1)^{n_1} \cdots (l_{k-1})^{n_{k-1}} p_k(l_k)^\omega \in \Gamma^\omega \mid (n_1, \ldots, n_{k-1}) \models \phi(\mathtt{y}_1, \ldots, \mathtt{y}_{k-1}) \text{ and } n_1, n_2, \ldots, n_k > 0\}$$

Note that the formula $\phi(\mathtt{y}_1, \ldots, \mathtt{y}_{k-1})$ used only $k-1$ variables because we do not specify the number of times the last loop l_k is taken since it is assumed to be traveled infinitely. The size

of cps, written $\texttt{size}(\texttt{cps})$, is equal to $2k + \texttt{len}(p_1 \natural_1 \cdots p_{k-1} l_{k-1} p_k l_k) + \texttt{size}(\phi(\texttt{y}_1, \ldots, \texttt{y}_{k-1}))$. In the sequel, whenever we will define a constrained path schema cps over a constrained alphabet (at, ag_n, Γ), we will assume w.l.o.g. that the size of Γ is linear in the size of cps (if it is not the case, one can remove the letters of Γ which do not appear in cps). For a run ρ from an Affine Counter System and a constrained path schema cps, we say that ρ respects cps iff there exists a word $w \in L(\texttt{cps})$ such that $\rho \models w$.

4.4.2 *Decision Problems over Constrained Path Schemas*

We introduce here three decision problems on constrained path schemas that will pave the way towards obtaining upper bound for the model-checking of linear time properties over flat Translating Counter Systems. First, observe that, in general, constrained path schemas are defined under constrained alphabet (at, ag_n, Γ) with $\Gamma \subseteq 2^{at \cup ag_n}$. This in turn allows us to define all the decision problems we consider assuming that the specifications are over the constrained alphabet $(at, ag_n, 2^{at \cup ag_n})$ unless stated otherwise.

The first problem amounts to check whether the language defined by a constrained path schema is empty or not. We call it the *consistency problem* and define it as follows:

CONSISTENCY	
Input:	A constrained path schema cps;
Question:	Do we have $L(\texttt{cps}) \neq \emptyset$?

Note that CONSISTENCY amounts to check that the second argument of the constrained path schema is satisfiable. Let us first recall simple consequences of the classical result [Pot91, Corollary 1] which will be useful to state the complexity of the consistency problem but also useful later.

Theorem 4.1. *[Pot91] There exist polynomials* $\texttt{pol}_1(\cdot)$, $\texttt{pol}_2(\cdot)$ *and* $\texttt{pol}_3(\cdot)$ *such that for every guard* g *in* $\texttt{G}(\texttt{C}_n)$ *of size* N*, we have:*

(I) there exist $B \subseteq [0, 2^{\texttt{pol}_1(N)}]^n$ *and* $\boldsymbol{P}_1, \ldots, \boldsymbol{P}_\alpha \in [0.2^{\texttt{pol}_1(N)}]^n$ *with* $\alpha \leq 2^{\texttt{pol}_2(N)}$ *such that for every* $\boldsymbol{y} \in \mathbb{N}^n$, $\boldsymbol{y} \models \texttt{g}$ *iff there are* $\boldsymbol{b} \in B$ *and* $\boldsymbol{a} \in \mathbb{N}^\alpha$ *such that* $\boldsymbol{y} = \boldsymbol{b} + \boldsymbol{a}[1]\boldsymbol{P}_1 + \cdots + \boldsymbol{a}[\alpha]\boldsymbol{P}_\alpha$*;*

(II) if g *is satisfiable, then there is* $\boldsymbol{y} \in [0, 2^{\texttt{pol}_3(N)}]^n$ *such that* $\boldsymbol{y} \models \texttt{g}$*.*

Note that (II) is an immediate consequence of (I). Now, we can obtain the NP upper bound for the consistency problem (the lower bound being obtained directly by reducing SAT for instance). Indeed, a constrained path schema defines a non-empty language iff its formula can be satisfied by a tuple in $[0, 2^{\texttt{pol}_3(N)}]^{k-1}$ where N is its size. This allows us to state the next lemma.

Lemma 4.1. CONSISTENCY *is* NP*-complete.*

Another problem of interest is the *intersection non-emptiness problem for the specification language* $\mathcal{L}$. It is later shown to be related to model-checking problem. This problem is parameterized by a specification language $\mathcal{L}$ over a constrained alphabet (at, ag_n, Γ) and is defined as follows:

INTERSECTION-NONEMPTINESS($\mathcal{L}$)	
Input:	a constrained path schema cps, and a specification A from $\mathcal{L}$;
Question:	Is $L(\mathtt{cps}) \cap L(A) \neq \emptyset$?

As we shall see, for the specification languages $\mathcal{L}$ we consider (first-order logic, Büchi specifications, etc.), given a path schema $\mathtt{cps} = (p_1(l_1)^+ \cdots p_{k-1}(l_{k-1})^+ p_k(l_k)^\omega, \phi(\mathsf{y}_1, \ldots, \mathsf{y}_{k-1}))$ and a specification $A \in \mathcal{L}$ it is possible to establish a bound (at most exponential) B such that whenever $L(\mathtt{cps}) \cap L(A) \neq \emptyset$ there is a witness word $p_1(l_1)^{n_1} \cdots p_{k-1}(l_{k-1})^{n_{k-1}} p_k(l_k)^\omega$ belonging to the intersection and for which each n_i is bounded by B. Hence a way to solve an instance of the intersection non-emptiness problem is to guess $n_1, \ldots, n_{k-1}$ bounded by the corresponding bound B, and then to test that $p_1(l_1)^{n_1} \cdots p_{k-1}(l_{k-1})^{n_{k-1}} p_k(l_k)^\omega$ indeed belongs to $L(\mathtt{cps}) \cap L(A)$. This motivates the introduction of the last decision problem over constrained path schemas. This is the *membership problem for a specification language* $\mathcal{L}$ and constrained path schemas:

MEMBERSHIP($\mathcal{L}$)	
Input:	A constrained path schema $\mathtt{cps} = (p_1(l_1)^+ \cdots p_{k-1}(l_{k-1})^+ p_k(l_k)^\omega, \phi(\mathsf{y}_1, \ldots, \mathsf{y}_{k-1}))$, a specification A from $\mathcal{L}$, and $n_1, \ldots, n_{k-1} \in \mathbb{N}$;
Question:	Does $p_1(l_1)^{n_1} \cdots p_{k-1}(l_{k-1})^{n_{k-1}} p_k(l_k)^\omega \in L(\mathtt{cps}) \cap L(A)$?

We point out that here the n_i's are understood to be encoded in binary. Also note that the problem whether $p_1(l_1)^{n_1} \cdots p_{k-1}(l_{k-1})^{n_{k-1}} p_k(l_k)^\omega \in L(\mathtt{cps})$ is easy to solve since it amounts to check whether $(n_1, \ldots, n_{k-1}) \models \phi(\mathsf{y}_1, \ldots, \mathsf{y}_{k-1})$. Since both $(n_1, \ldots, n_{k-1})$ and $\phi(\mathsf{y}_1, \ldots, \mathsf{y}_{k-1})$ are provided as input, the check amounts to perform multiplication, addition and comparison of polynomially many bits, which can be done in polynomial time.

4.4.3 *Reduction from Intersection Non-Emptiness to Membership*

We explain here a property regarding a specification language $\mathcal{L}$ which allows to reduce (efficiently) the intersection non-emptiness problem to the membership problem. A specification language $\mathcal{L}$ is said to have an *exponentially bounded limiting loops map* if there exists a map $f_{\mathcal{L}}$, which takes as argument a specification in $\mathcal{L}$ and a constrained path schema, and a polynomial $\mathtt{pol}(\cdot)$ such that for all $A \in \mathcal{L}$ and all constrained path schemas $\mathtt{cps} = (p_1(l_1)^* \cdots p_{k-1}(l_{k-1})^* p_k(l_k)^\omega, \phi(\mathsf{y}_1, \ldots, \mathsf{y}_{k-1}))$, we have the two following properties:

1. $f_{\mathcal{L}}(A, \mathtt{cps})$ is computable and $f_{\mathcal{L}}(A, \mathtt{cps}) \leq 2^{\mathtt{pol}(\mathtt{size}(A)+\mathtt{size}(\mathtt{cps}))}$;
2. $L(\mathtt{cps}) \cap L(A) \neq \emptyset$ if and only if there is an infinite word $p(l_1)^{n_1} \cdots l_{k-1}(l_{k-1})^{n_{k-1}} p_k(l_k)^\omega$ in $L(\mathtt{cps}) \cap L(A)$ verifying $n_i \leq f_{\mathcal{L}}(A, \mathtt{cps})$ for all $i \in [1, k-1]$.

Consequently a way to solve an instance of the intersection non-emptiness problem for specification languages having an exponentially bounded limiting loops map con-

sists in guessing $n_1, \ldots, n_{k-1}$ bounded by $f_{\mathcal{L}}(A, \mathtt{cps})$ and then to test that the word $p(l_1)^{n_1} \cdots l_{k-1}(l_{k-1})^{n_{k-1}} p_k(l_k)^{\omega}$ indeed belongs to $L(\mathtt{cps}) \cap L(A)$. This observation leads to the following proposition which makes a connection between the Intersection Non-Emptiness problem and the Membership problem. Before we state the lemma, let us define two generalised complexity classes. This is necessary since, in the next part we will talk about a range of complexity classes starting from PTIME to PSPACE. We consider $\mathcal{C}$ as a class of Turing machines characterizing a complexity class including PTIME. Also, let $\mathcal{NC}$ be its corresponding non-deterministic class of Turing machines. In other words if we take $\mathcal{C}$ to be equal to the class PTIME (or PSPACE) then $\mathcal{NC}$ corresponds to exactly the class NP (NPSPACE respectively).

Proposition 4.1. *[DDS13] If $\mathcal{L}$ is a specification language having an exponentially bounded limiting loops map and such that its membership problem is in $\mathcal{C}$, then its intersection non-Emptiness problem is in $\mathcal{NC}$.*

4.4.4 *Reduction From Model-Checking to Intersection Non-Emptiness*

We present here the main property that allows us to abstract runs of a flat Translating Counter Systems into a finite set of constrained path schemas. The main ingredients to obtain this result are detailed in [DDS15]. The first idea is that since the system is flat, we can enumerate an exponential number of minimal path schemas, which are a succession of simple path and simple loops in the counter system where no transition occurs more than twice. We know then that any run follows necessarily the structure of one of this minimal path schema. To go from a minimal path schema to a constrained path schema, it remains to change the visited control state into labels belonging to the set of atomic propositions at and the set of atomic guards ag_n. The main difficulty here lies in the fact that when a run goes through a loop of a minimal path schema, the set of atomic guards that are satisfied might evolve while taking the loop many times, but as shown in [DDS15, Theorem 7.11], when dealing with Translating Counter Systems it is possible to unfold (at most a polynomial number of times) such loops in order to obtain a set of constrained path schemas which cover all the runs going through a minimal path schema. Such a construction is possible because in a Translating Counter System, when a loop is taken many times, for each counter the behavior is monotone and hence one can follow easily how the satisfiability of the different atomic guards evolve. For instance, if we have a loop that increases a counter x_1 and three atomic guards of the form $\mathrm{x}_1 < 2$, $\mathrm{x}_1 > 3$ and $\mathrm{x}_1 > 5$, then we know that these guards will be verified exactly in these order while taking the loop. This reasonning can be adapted to more complex guards and to the case where the updates are performed on many counters. Furthermore a last argument is necessary in order for this unfolding to be correct. In a Translating Counter System if an atomic guard of the form $\sum_{i=1}^{n} a_i \cdot \mathrm{x}_i \sim b$ is satisfied in the first and last configurations of a sequence of transition, then by a convexity argument, this guard is satisfied by all the configurations along the sequence of transitions. These different properties lead us to the following proposition.

Proposition 4.2. *[DDS15; DDS13] Let at be a finite set of atomic propositions, ag_n be a finite set of atomic guards from $\mathtt{G}(\mathtt{C}_n)$, S be a flat Translating Counter System, whose atomic propositions and atomic guards are from $at \cup ag_n$, and $c_0 = (q_0, \boldsymbol{v}_0)$ be an initial configuration. One can construct in exponential time a set X of constrained path schemas such that:*

- *Each constrained path schema* cps *in* X *has an alphabet of the form* (at, ag_n, Γ) (Γ *may vary) and* cps *is of polynomial size in* $\texttt{size}(S) + \texttt{size}(c_0) + \texttt{size}((at, ag_n, \emptyset))$.
- *Checking whether a constrained path schema belongs to* X *can be done in polynomial time.*
- *For every run* ρ *from* c_0*, there exists a constrained path schema* cps *in* X *and a word* $w \in L(\texttt{cps})$ *such that* $\rho \models w$.
- *For every constrained path schema* cps *in* X *and for every* $w \in L(\texttt{cps})$*, there is a run* ρ *from* c_0 *such that* $\rho \models w$.

This last proposition allows to reduce the model-checking problem for flat Translating Counter System to the intersection non-emptiness problem for specification language $\mathcal{L}$ over $(at, ag_n, 2^{at \cup ag_n})$. The idea of this reduction works as follows: given a flat Translating Counter System S, an initial configuration c_0 and a specification $A \in \mathcal{L}$, a way to solve the model-checking problem is to guess in polynomial time a constrained path schema cps from the set X and then to solve $L(\texttt{cps}) \cap L(A) \neq \emptyset$. Proposition 4.2 ensures the correctness of this non-deterministic procedure and also that the guess and checking of cps can both be done in polynomial time.

Proposition 4.3. *[DDS13] If the intersection non-emptiness problem for a specification language* $\mathcal{L}$ *is in* $\mathcal{C}$*, then* CS-Model-Checking$(\mathcal{L})$ *over flat Translating Counter System is in* $\mathcal{NC}$.

4.4.5 *General algorithm*

By combining the results presented in Proposition 4.1 and Proposition 4.3, we deduce a general reduction from the model-checking problem over flat Translating Counter System of a specification language $\mathcal{L}$ to the membership problem for $\mathcal{L}$ when the $\mathcal{L}$ has an exponentially bounded limiting loops map.

Theorem 4.2. *If* $\mathcal{L}$ *is a linear specification language having an exponentially bounded limiting loops map and such that its membership problem is in* $\mathcal{C}$*, then* CS-Model-Checking$(\mathcal{L})$ *over flat Translating Counter System is in* $\mathcal{NC}$.

Hence the roadmap we follow in [DDS13] to obtain complexity bounds for the model-checking problem over flat Translating Counter Systems with various specification languages consist in proving that the considered languages admit an exponentially bounded limiting loop map and also an associated membership problem either in Ptime or in Pspace.

Finally, from the proofs of Proposition 4.1 and Proposition 4.3 we can extract the general non-deterministic Algorithm 1 to solve the model-checking problem of flat counter systems with languages specification having an exponentially bounded limiting loops.

4.4.6 *An illustrative example*

In this section, we give an example of the concepts and constructions encountered in the previous sections. In particular we show how a constrained path schema is obtained from a given flat counter system and a specification as explained in Proposition 4.2. We start with a flat Translating Counter System S and a formula ϕ in a specification language $\mathcal{L}$, that is satisfied by S. We will then follow the Algorithm 1 to guess a path schema cps with a constrained alphabet, from S such that there is a run ρ in S satisfying ϕ iff there exists a word $w \in L(\texttt{cps}) \cap L(\phi)$.

Algorithm 1 Solving CS-Model-Checking($\mathcal{L}$) for $\mathcal{L}$ with an exponentially bounded limiting loop map $f_{\mathcal{L}}$ over flat Translating Counter Systems

Input: A flat Translating Counter System S
Input: A configuration c_0
Input: A specification A from $\mathcal{L}$
Output: Is there a run ρ in S starting from c_0 such that $\rho \models A$?

1: Guess $\mathtt{cps} = (p_1(l_1)^* \cdots p_{k-1}(l_{k-1})^* p_k(l_k)^\omega, \phi(\mathrm{y}_1, \ldots, \mathrm{y}_{k-1}))$ in X [see Proposition 4.2]
2: Guess $\mathbf{y} \in [0, f_{\mathcal{L}}(A, \mathtt{cps})]^{k-1}$
3: **if** $p_1(l_1)^{\mathbf{y}[1]} \cdots p_{k-1}(l_{k-1})^{\mathbf{y}[k-1]} p_k(l_k)^\omega \in L(A) \cap L(\mathtt{cps})$ **then**
4: Return **True**
5: **else**
6: Return **False**
7: **end if**

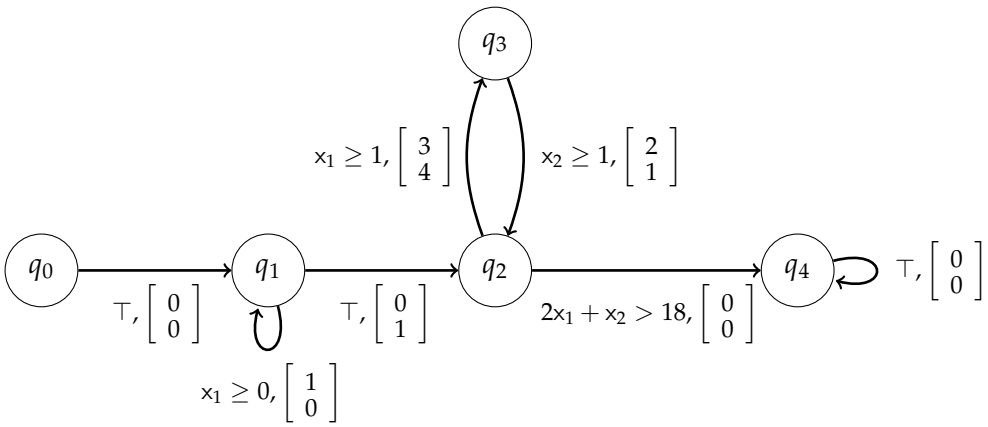

Figure 4.4: A flat Translating Counter System

We consider the flat Translating Counter System S of dimension 2 as shown in Figure 4.4 with the initial configuration $c_0 = (q_0, [0,0])$ and the specification $\psi = \mu z_1.(\nu z_2.(p \wedge (x_1 - x_2 = 0) \wedge \mathtt{X}\mathtt{X}z_2) \vee \mathtt{X}z_1)$ in linear μ-calculus. The specification essentially states that in a run there exists a reachable position after which every even position has same value in both counters and the label p holds. The labelling function associated with S is $\mathbf{l}(q_4) = \{p\}$ and $\mathbf{l}(q) = \{r\}$ for $q \neq q_4$. As explained, we guess then a constrained path schema over a suitable constrained alphabet from X. For the constrained alphabet, for this case, we get that $at = \{\mathtt{p}, r\}$ and $ag_2 = \{x_1 \geq 0, x_2 \geq 1, x_1 \geq 1, 2x_1 + x_2 > 18, x_1 - x_2 = 0\}$. It remains to define Γ in (at, ag_n, Γ) where $\Gamma \subseteq 2^{at \cup ag_n}$. As explained before, our aim is to label each state in the constrained path schema with the guards and atomic propositions that are satisfied at this instant. Hence, we include in Γ only those subsets of $at \cup ag_2$ that are useful in labelling the states of the constrained path schema. Since the guards satisfied at any state in a loop in a path schema can change for different iterations, we might have to unfold the loops to label them with correct set from Γ consistently. To check that any run respecting the labelled path schema stays in a loop only as long as the counter values are consistent with the labels of states of the loop, we have a formula ϕ characterizing the number of times each loop can be taken. This allows us to remove the guards and updates from the counter systems and gives us a set X of constrained path schemas as described in Proposition 4.2.

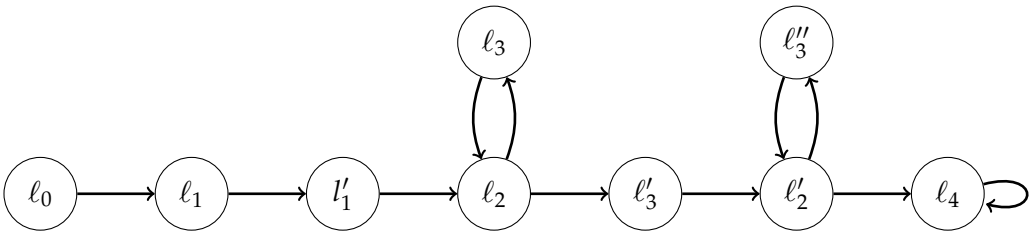

Figure 4.5: A Constrained Path Schema for the flat Translating Counter System of Figure 4.4

Figure 4.5 shows the structure of a constrained path schema in X obtained from S. To obtain this structure we have unfold once the loop on q_1 which gives us the path between ℓ_1 and ℓ_1' and we have divided into two loops the loop between the set q_2 and q_3 to obtain one loop between ℓ_2 and ℓ_3 and one between ℓ_2' and ℓ_3'''. To ease the understanding each subscript corresponds to the number of the associated control state, for instance ℓ_1 and ℓ_1' are associated with the control state q_1. The different subsets of $2^{at \cup ag_2}$ we used in this representation are described below:

- ℓ_0 stands for $\{r, x_1 \geq 0, x_1 - x_2 = 0\}$,
- ℓ_1 stands for $\{r, x_1 \geq 0, x_1 - x_2 = 0\}$,
- ℓ_1' stands for $\{r, x_1 \geq 0, x_1 \geq 1\}$,
- ℓ_2 stands for $\{r, x_1 \geq 0, x_1 \geq 1, x_2 \geq 1, x_1 - x_2 = 0\}$,
- ℓ_3 stands for $\{r, x_1 \geq 0, x_1 \geq 1, x_2 \geq 1\}$
- ℓ_3' stands for $\{r, x_1 \geq 0, x_1 \geq 1, x_2 \geq 1, 2x_1 + x_2 > 18\}$,
- ℓ_2' stands for $\{r, x_1 \geq 0, x_1 \geq 1, x_2 \geq 1, x_1 - x_2 = 0, 2x_1 + x_2 > 18\}$,
- ℓ_3''' stands for $\{r, x_1 \geq 0, x_1 \geq 1, x_2 \geq 1, 2x_1 + x_2 > 18\}$,

- ℓ_4 stands for $\{p, x_1 \geq 0, x_1 \geq 1, x_2 \geq 1, x_1 - x_2 = 0, 2x_1 + x_2 > 18\}$.

As a matter of fact, the constrained alphabet associated to this constrained path schema is such that $\Gamma = \ell_0 \cup \ell_1 \cup \ell_1' \cup \ell_2 \cup \ell_3 \cup \ell_2' \cup \ell_3' \cup \ell_3'' \cup \ell_4$. As noted earlier, Γ is a subset of $2^{at \cup ag_2}$ but is of polynomial in $\texttt{size}(S) + \texttt{size}(c_0) + size(at, ag_2, \emptyset)$. It remains to provide the formula $\phi(y_1, y_2)$ stating how many times the two first loops of this constrained path schema can be travelled. For example, note that following the updates in S, we must take the first loop exactly once, to preserve the labels (if we take more times this loop, then the labels $2x_1 + x_2 > 18$ will in fact become true and we will have to jump to another loop). For what concerns the second loop it can be taken any number of times and the validity of the encountered labels will still be ensured. Hence we deduce $\phi(y_1, y_2) = y_1 = 1 \wedge y_2 \geq 1$ (recall that in a constrained path schema, by definition any loop is taken at lest once). If we call cps this constrained path schema, then it remains to check that $L(\texttt{cps}) \cap L(\psi) \neq \emptyset$ (which here clearly holds).

4.5 RESULTS FOR FLAT TRANSLATING COUNTER SYSTEMS

In [DDS13], by applying the general roadmap given by Propositions 4.1 and 4.3 and Theorem 4.2, we were able to obtain upper bounds for the model-checking problem over flat Translating Counter Systems of different specification languages presented in Section 4.1.

The first lower bound we obtained was for the specification language Past LTL in [DDS12; DDS15]. In this latter work, we did not follow exactly the roadmap provided in the previous section, because we developed a technique specific to Past LTL, however it appears that the same results could be obtained by applying Theorem 4.2. First, we obtain a exponential limiting loops map for Past LTL [DDS15, Theorem 4.1] by showing that Past LTL enjoys some stuttering invariants. More precisely, given a constrained path schema $\texttt{cps} = (p_1(l_1)^+ \cdots p_{k-1}(l_{k-1})^+ p_k(l_k)^\omega, \phi(y_1, \ldots, y_{k-1}))$, a formula ψ in Past LTL and $n_1, \ldots, n_{k-1} \in \mathbb{N}$, we show that if $p_1(l_1)^{n_1} \cdots p_{k-1}(l_{k-1})^{n_{k-1}} p_k(l_k)^\omega \models \psi$ and if n_i is bigger than $2.td(\psi) + 5$ (where td corresponds to the temporal depth of ψ), then we can replace n_i by any n_i' bigger than $2.td(\psi) + 5$ and the obtained path will still satisfy the formula ψ. Furthermore, we know from [MS03], that there is PTIME procedure to check the Past LTL formula ψ over an ultimate periodic path of the form $u(w)^\omega$ (where u and w are finite paths). This allows us to conclude that the membership problem for Past LTL is in PTIME. Using Theorem 4.2, we conclude that the model-checking of Past LTL over flat Translating Counter System is in NP.

In [DDS12; DDS15] we obtain as well matching lower bounds, for this we provide different hardness result. First we know from [KF11] that the model-checking of Past LTL over flat Kripke structures is NP-complete hence this hardness result still holds for flat Translating Counter System. But we show as well that CS-REACH is NP-hard for flat Translating Counter System and since we can encode CS-REACH as a model-checking problem using Past LTL as a specification language, this provides another proof for the lower bound.

Theorem 4.3. *[DDS15]* CS-REACH *and* CS-MODEL-CHECKING(*Past LTL*) *for flat Translating Counter Systems are* NP*-complete.*

In [DDS13], we first show that the specification language given by Büchi automata over the alphabet $\Sigma = 2^{at \cup ag_n}$ admits an exponentially bounded limiting loops map. For this

we prove a specific stuttering theorem on Büchi automata and we use the fact that for a constrained path schema $\mathrm{cps} = (p_1(l_1)^+ \cdots p_{k-1}(l_{k-1})^+ p_k(l_k)^\omega, \phi(y_1, \ldots, y_{k-1}))$, the set $[\![\phi(y_1, \ldots, y_{k-1})]\!]$ is semi-linear and hence somehow ultimately periodic. We furthermore show that for this language, the membership problem is in PTIME. We obtain this result by encoding the ultimately periodic words by straight-line programs and by observing that checking whether an ultimately periodic word given by a straight-line program is recognized by a Büchi automaton can be done in PTIME [MS03] . This allows us to obtain results for specification in BS. We recall that specification in BS are automata labelled with boolean combination over $at \cup ag_n$. Any specification in BS can be translated into a Büchi automaton over the alphabet $2^{at \cup ag_n}$ but this might lead to an exponential blowup. However in our context we can avoid the blowup since we consider constrained path schema over a constrained alphabet (at, ag_n, Γ) where the size of Γ is linear in the size of the constrained path schema. Hence to obtain the desired upper bound we construct from a specification a BS, a Büchi automaton over the alphabet Γ. Thanks to this trick, we get that BS admits an exponentially bounded limiting loops map and that its membership problem is in PTIME. Using Theorem 4.2 we get an upper bound for the model-checking of BS over flat Translating Counter Systems and the matching lower bound is obtained by reducing the reachability problem.

Theorem 4.4. *[DDS15]* CS-MODEL-CHECKING(*BS*) *over flat Translating Counter Systems is* NP*-complete.*

For the specification language ABS and the linear μ-calculus, the treatment is similar. In fact, we provide a PSPACE upper bound for any specification languages having the nice Büchi property. A specification language $\mathcal{L}$ is said to have the nice Büchi property if any specification $A \in \mathcal{L}$ can be translated into a Büchi automaton (which accepts the same language) such that the size of each state of the automaton is polynomial in the size of A and such that it can be checked in polynomial time if a state is initial and such that the transition relation can be decided in polynomial space. We prove then that if a specification language has the nice Büchi property then it has an exponentially bounded limiting loops map (we use the previous result on Büchi automaton) and its membership problem is in PSPACE (to solve this problem we check on the fly whether an ultimately periodic word is accepted by the Büchi automaton associated to the specification, which avoids to build the automaton). Using known results on alternating Büchi automata [MH84] and on linear μ-calculus [Var88], we deduce that ABS and the linear μ-calculus have the nice Büchi property. To obtain a matching PSPACE lower bound for these two specification languages, we use a reduction from the nonemptiness of a finite alternating automaton over a singleton alphabet [JS07].

Theorem 4.5. *[DDS15]* CS-MODEL-CHECKING(*ABS*) *and* CS-MODEL-CHECKING(*linear μ-calculus*) *over flat Translating Counter Systems are* PSPACE*-complete.*

Finally for FO, we show as well by establishing a stuttering theorem that it has an exponentially bounded limiting loop maps. We know as well that checking whether an ulimately periodic word satisfies a first order logic formula can be done in PSPACE [MS03], however due to the succinct encoding of the repetitions of loops in the membership problem, this result will lead to an EXPSPACE upper bound for this latter problem. We are able to

obtain a PSPACE algorithm by using a technique which avoids to build explicitely the word corresponding to the unfolding of the constrained path schema. The lower bound for the model-checking problem for FO comes from the fact that checking whether an utlimately periodic path satisfies a first order formula is PSPACE-hard.

Theorem 4.6. *[DDS15]* CS-MODEL-CHECKING(FO) *over flat Translating Counter System is* PSPACE*-complete.*

Note that this last result is one of the most surprising since we know that model-checking first order logic over Kripke structures is non-elementary [Sto74] and here it is true that the underlying structure of the considered systems is simple but these systems manipulate counter whose values can be tested by atomic propositions in the logic.

The Table 4.1 sums up the results we have obtained for the model-checking of flat Translating Counter System. We point out that most of our lower bounds are true without considering counters which lead to new results as well for the model-checking of flat Kripke Structures. In the last column of this table, we recall as well the complexity of the model-checking problem for the considered specification languages over classical Kripke structures.

	flat $\mathcal{TCS}$	flat $\mathcal{KS}$	$\mathcal{KS}$
BS	NP-complete [Thm 4.4]	in PTIME	in PTIME
ABS	PSPACE-complete [Thm 4.5]	PSPACE-complete [Thm 4.5]	PSPACE-complete
Past LTL	NP-complete [Thm 4.3]	NP-complete [Thm 4.3], [KF11]	PSPACE-complete [SC85]
linear μ-calculus	PSPACE-complete [Thm 4.5]	PSPACE-complete [Thm 4.5]	PSPACE-complete [Var88]
FO	PSPACE-complete [Thm 4.6]	PSPACE-complete [Thm 4.6]	Non-elementary [Sto74]

Table 4.1: Complexity of the model-checking problem

4.6 DEALING WITH AFFINE COUNTER SYSTEMS

As we have explained, for flat Affine Counter Systems with the finite monoid property, it has been shown in [FL02] that the reachability problem is decidable and in [Dem+10] that the decidability still hold for the model-checking of an extension of the branching time temporal logic CTL* (which can express properties in LTL). For these two problems, the resolution technique consists in a reduction to the satisfiability of a Presburger formula which is exponential in the size of the system, hence this leads to decision procedures in 4EXPTIME. In [IS16], inspired by our previous works on flat Translating Counter Systems, we looked whether these complexity bounds could be improved and provide algorithms

with much better complexity for the reachability problem and the model-checking of Past LTL and FO but this time the atomic propositions cannot refer to the values of the counters in the systems.

4.6.1 *Hardness result for the control state reachability*

First, we show that the control state reachability problem for flat Affine Counter Systems with the finite monoid property is Σ_2^P-hard. We recall that Σ_2^P corresponds to the class $\mathrm{NP}^{\mathrm{NP}}$ in the polynomial hierarchy, i.e. the problems that can be solved in NP by a Türing machine using an oracle in NP. To show this result, we perform a reduction from the validity problem for the $\exists^*\forall^*$ fragment of quantified boolean formulas (Σ_2-QBF), which is a well-known Σ_2^P-complete problem (see for instance [ABo09]). We present here the reduction only to give an hint on what can be achieved with flat Affine Counter Systems with the finite monoid property. Let us consider a formula $\phi := \exists y_1 \dots \exists y_p \forall z_1 \dots \forall z_r \ . \ \psi(y_1, \dots, y_p, z_1, \dots, z_q)$, where $\{y_1, \dots, y_p, z_1, \dots, z_r\}$ are non-empty sets of boolean variables, and ψ is a quantifier-free boolean formula. We build, in polynomial time, a flat Affine Counter System S_ϕ, with the finite monoid property, such that Φ is valid if and only if S_ϕ has a run reaching the control state q_f which starts in $(q_0, \mathbf{v}_0)$ for a certain valuation $\mathbf{v}_0$ of its counters.

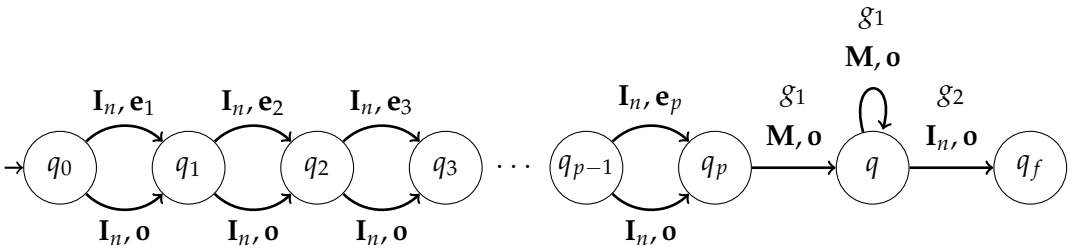

Figure 4.6: The counter system S_ϕ corresponding to the Σ_2-QBF ϕ

Let π_n denote the n-th prime number, i.e. $\pi_1 = 2, \pi_2 = 3, \pi_3 = 5$, etc. Formally, $S_\phi = (Q, \mathtt{C}_n, \Delta, \mathbf{l})$, where $Q = \{q_0, \dots, q_p, q, q_f\}$, $n = p + \sum_{k=1}^{r} \pi_j$, and $\mathbf{l}$ is the function associating to each state an empty set of propositions (we do not consider any label here since we are interesting in the reachability problem). We recall that π_k is polynomial in the size of k, hence n is as well polynomial in the size of ϕ. The transition rules Δ are depicted in Figure 4.6. Intuitively, each existentially quantified boolean variable y_i of ϕ is modeled by the counter x_i in S_ϕ, each universally quantified variable z_j of ϕ is modeled by the counter $x_{p+\sum_{k=1}^{j} \pi_k}$, and the rest are working counters. All counters range over the set $\{0, 1\}$, with the obvious meaning (0 stands for false and 1 for true).

The counter system S_ϕ works then in two phases. The first phase, corresponding to transitions $q_0 \to \dots \to q_p$, initialises the counters $x_1, \dots, x_p$ to some values from the set $\{0, 1\}$, thus mimicking a choice of boolean values for the existentially quantified variables $y_1, \dots, y_p$ from ϕ. We recall that $\mathbf{I}_n \in \mathbb{Z}^{n \times n}$ is the identity matrix, and $\mathbf{e}_k \in \{0, 1\}^n$ is the unit vector such that $\mathbf{e}_k[i] = 0$ if $i \neq k$ and $\mathbf{e}_k[i] = 1$.

The second phase checks that ϕ is valid for each choice of $z_1, \dots, z_r$. This is done by the loop on the control state q, which explores all combinations of 0's and 1's for the counters $x_{p+\sum_{k=1}^{j} \pi_k}$, corresponding to z_j, for all $j \in [1, r]$. To this end, we use the permutation matrix $\mathbf{M}$, which consists of $\mathbf{I}_p$ and r rotation blocks $\mathbf{M}_{\pi_j} \in \{0, 1\}^{\pi_j \times \pi_j}$ (see Figure 4.7). The valuation $\mathbf{v}_0$ ensures that the initial value of $x_{p+\sum_{n=1}^{j} \pi_n}$ is 1, for all $j \in [1, r]$, the other

counters being 0 initially. All the guards in the system are set to true except the two guards g_1 and g_2. Intuitively, after n iterations of the affine function $(\mathbf{M}, \mathbf{o})$, labeling the loop on q in S_Φ, we have $x_{p+\sum_{k=1}^{j}\pi_k} = 1$ iff n is a multiple of π_j. This fact guarantees that all combinations of 0's and 1's for $z_1, \ldots, z_r$ have been visited in $\Pi_{j=1}^{q}\pi_j$ iterations of the cycle. The guard g_1, labeling the cycle, tests that, at each iteration, the formula ψ is satisfied, using a standard encoding of the formula ψ. Namely, each variable y_i is encoded as the term $x_i \geq 1$ and each z_j is encoded as $x_{p+\sum_{k=1}^{j}\pi_k} \geq 1$. For instance, the formula $y_1 \vee \neg z_2$ is encoded as $x_1 \geq 1 \vee \neg(x_{p+\pi_1+\pi_2} \geq 1))$ which is equivalent to $x_1 \geq 1 \vee x_{p+\pi_1+\pi_2} < 1$. Finally, the guard g_2 simply checks that $x_{\pi_1} = \ldots = x_{\pi_1+\ldots+\pi_r s} = 1$, ensuring that the loop has been iterated sufficiently many times. This construction allows us to deduce that q_f is reachable in S_ϕ if and only if ϕ is satisfiable.

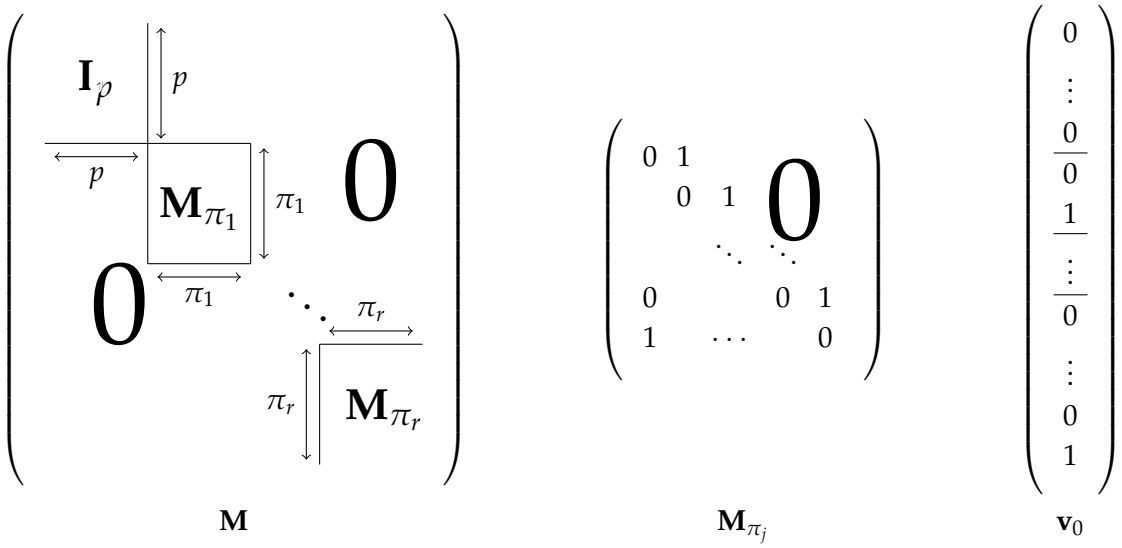

Figure 4.7: Matrix $\mathbf{M}$ and initial vector $\mathbf{v}_0$

Finally, it is clear that the Affine Counter System S_ϕ is flat and one can easily check that it has the finite monoid property. This leads to the desire hardness result.

Theorem 4.7. CS-ControlReach *and* CS-Reach *are* Σ_2^P*-hard for flat Affine Counter Systems with the finite monoid property.*

4.6.2 *Bounding the iterations of cycles*

We present now the technique that allows us to get upper bound for the reachability problem when considering flat Affine Counter Systems with the finite monoid property. We use a methodology similar to the one presented in Section 4.4, however we consider some restrictions to ease the reasoning. Indeed we assume that the guards in the counter systems do not have disjunctions. Formally, an Affine Counter Systems $S = (Q, \mathtt{C}_n, \Delta, \mathbf{l})$ is said to be disjunction free if for all edges $\delta \in \Delta$, we have that $guard(\delta)$ is a conjunction of atomic guards of the form $\sum_{i=1}^{n} a_i \cdot \mathtt{x}_i \sim b$ where the a_i's are in $\mathbb{Z}$, $b \in \mathbb{N}$ and $\sim \in \{=, \leq, \geq, <, >\}$. Thanks to this restriction, we know that the set of valuations which satisfy a guard is a

convex set. This allows us to avoid to track in the unfolding of our systems the atomic guards that are valid as it is done in Theorem 4.2 for Translating Counter Systems.

We know explain the techniques used in [IS16] to solve the reachability problem for a disjunction free flat Affine Counter Systems with the finite monoid property $S = (Q, \mathtt{C}_n, \Delta, \mathbf{l})$. Instead of using constrained path schemas as presented in the previous section, we come back to using simple path schemas as defined in [DDS15] for our first result on the model-checking of Past LTL over flat Translating Counter Systems. Formally, a *path schema* is a non-empty finite sequence $P := u_0 \dots u_N$, where u_i is either an edge from Δ or a simple cycle in S, such that (1) $u_0, \dots, u_N$ are pairwise distinct, (2) u_N is a simple cycle and (3) $u_0 \dots u_N$ corresponds to a path in S. We know from [DDS15] that the number of such path schemas is finite and can be exponential in the size of S. In order to represent runs of S thanks to path schema, we have to iterate all the loops a finite number of times and the last one is taken infinitely. An *iterated path schema* is hence a pair $(P, \mathbf{m})$, such that P is a path schema, and $\mathbf{m} \in \mathbb{N}^{\mathtt{len}(P)-1}$ is a vector, where $\mathbf{m}[i+1] \geq 1$ and $\mathbf{m}[i+1] > 1$ implies that $P(i)$ is a cycle for all $i \in [0, \mathtt{len}(P) - 2]$. An iterated path schema defines a unique infinite word over Δ, denoted by $edges(P, \mathbf{m}) = P(0)^{\mathbf{m}[1]} P(1)^{\mathbf{m}[2]} \cdots P(\mathtt{len}(P) - 2)^{\mathbf{m}[\mathtt{len}(P)-1]} P(\mathtt{len}(P))^{\omega}$.

In order to match runs and iterated path schemas, for an infinite run $\rho := c_0 \xrightarrow{\delta_0} \cdots \xrightarrow{\delta_{m-1}} c_m \xrightarrow{\delta_m} \cdots$ of S, we denote by $edges(\rho)$ the infinite word $\delta_0\delta_1\delta_2 \dots$. Note that for a run, there could be more than one iterated path schema associated to it because of the way we could unfold some loops. For this reason we define $ips(\rho) = \{(P, \mathbf{m}) \mid (P, \mathbf{m}) \text{ is an iterated path schema and } edges(\rho) = edges(P, \mathbf{m})\}$. From [DDS15], we know that if ρ is a run then $ips(\rho) \neq \emptyset$, hence the iterated path schemas are suitable to represent all the runs.

For disjunction free flat Translating Counter Systems, the notion of iterated path schema suffices to solve reachability. The idea is that for each path schema, one can build a quantifier free Presburger formula whose free variables correspond to the number of times each simple cycles can be taken in order to have a run. There are two reasons which make this construction possible, first since the updates of the counter are translations, it is easy to know the effect of the iteration of a cycle on each counter, hence one can compute the value of the counters for each instantiation of the free variables. However this is not enough to obtain a run, because one needs to ensure that the guards are satisfied along this path in the counter system. But in a disjunction free system, since the guards define convex sets, it is enough for each iteration of a cycle to check that the guards along the cycle are satisfied in the first and in the last iteration. Once this observed, it is easy to build a quantifier free Presburger formula ϕ for each path schema P such that $\mathbf{m} \models \phi$ if and only if there is a run ρ verifying $edges(\rho) = edges(P, \mathbf{m})$.

When considering flat Affine Counter Systems with the finite monoid property, the used arguments are similar, but the reasoning needs to be refined because of the fact that the updates of the counters are much more complex. However we will see that in the end, it reduces to building a system of equations (similar to a quantifier free Presburger formula) whose variables correspond to the numbers of times the cycles of a path schema can be iterated. Thanks to a linear algebra result, we show that such a system of equations, even if it has an exponential number of lines admits a small solution, this allows to bound the number of times a loop is taken in an iterated path schema in order to have a witness run.

In the sequel we fix a run ρ of S starting at a configuration c_0, and an iterated path schema $(P, \mathbf{m}) \in ips(\rho)$. We consider a simple cycle $c = \delta_0 \dots \delta_{k-1}$ of P and we assume that $update(\delta_i) = (\mathbf{A}_i, \mathbf{b}_i)$ for all $i \in [0, k-1]$. We consider the affince function $f_c = (\mathbf{A}_c, \mathbf{b}_c)$

associated to the entire cycle c, where $\mathbf{A}_c = \mathbf{A}_{k-1} \cdots \mathbf{A}_1 \cdot \mathbf{A}_0$ and $\mathbf{b}_c = \sum_{i=0}^{k-1} \prod_{j=k-1}^{i+1} \mathbf{A}_j \cdot \mathbf{b}_i$. Since S has the finite monoid property, the set $\mathcal{M}_c = \{\mathbf{A}_c^i \mid i \in \mathbb{N}\}$ is finite. Then there exist two integer constants $\alpha, \beta \in \mathbb{N}$, such that $0 \leq \alpha + \beta \leq card(\mathcal{M}_c) + 1$, and $\mathbf{A}_c^\alpha = \mathbf{A}_c^{\alpha+\beta}$. Observe that, in this case, we have $\mathcal{M}_c = \{\mathbf{A}_c^0, \ldots, \mathbf{A}_c^\alpha, \ldots, \mathbf{A}_c^{\alpha+\beta-1}\}$.

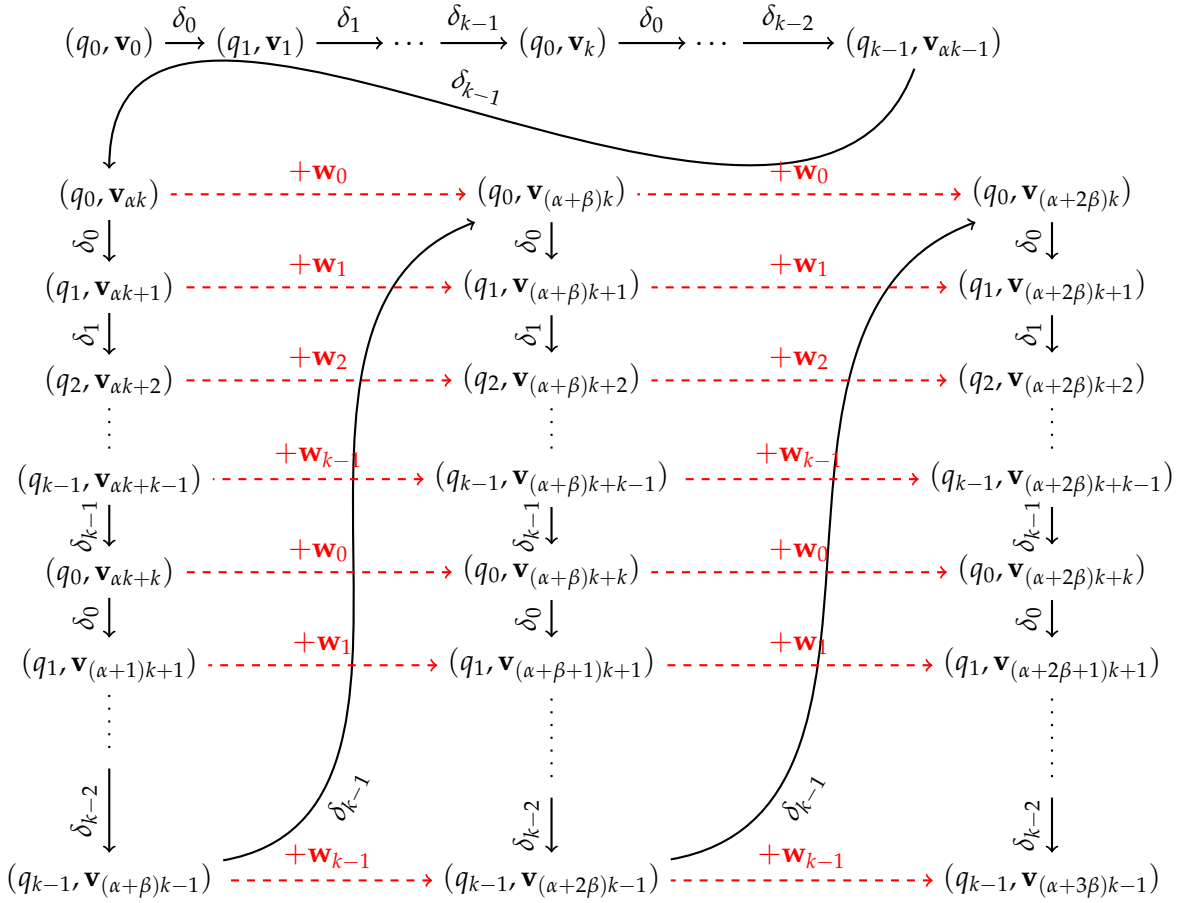

Figure 4.8: Behaviour of an execution which iterates $\alpha + 3\beta$ times the cycle $c = \delta_0 \ldots \delta_{k-1}$

We exhibit another run ρ' of S and an iterated path schema $(P, \mathbf{m}') \in ips(\rho')$, such that $\mathbf{m}'[i] \leq 2^{Poly(\mathtt{size}(S)+\mathtt{size}(c_0))}$ for all $i \in [1, \mathtt{len}(P) - 1]$, for a polynomial function $Poly(x)$. Because $c = \delta_0 \ldots \delta_{k-1}$ is a simple cycle of P and $(P, \mathbf{m}) \in ips(\rho)$, there exists a (possibly infinite) contiguous subsequence of ρ, let us call it $\theta = (q_0, \mathbf{v}_0) \xrightarrow{\tau_0} (q_1, \mathbf{v}_1) \xrightarrow{\tau_1} \ldots$ that iterates c, i.e. $\tau_i = \delta_{(i \bmod k)}$, for all $i \geq 0$. The main intuition now is that θ can be decomposed into a prefix of length $(\alpha + \beta)k$ and k infinite sequences of translations along some effectively computable vectors $\mathbf{w}_0, \ldots, \mathbf{w}_{k-1}$. More precisely, all valuations $\mathbf{v}_i$ of θ, for $i \geq (\alpha + \beta)k$, that are situated at distance βk one from another, differ by exactly the same vector. Figure 4.8 provides a graphical representation of this idea where we show that morally if the cycle is iterated more that $(\alpha + 2\beta)$ times then the different encountered configuration differ by a translation.

As a consequence, we can use a reasoning similar to the one explained above for Translating Counter System and deduce that the number of times each loop can be iterated to form a run is the solution of a big system of equations where the number of variables correspond to the number of loops in P. To show that such a system admits a small solution, we prove the

following result where $||\mathbf{v}|| = \max_{i=1}^{n} |\mathbf{v}[i]|$ for $\mathbf{v} \in \mathbb{Z}^n$ and $||\mathbf{A}|| = \max_{i=1}^{m} \max_{j=1}^{n} |\mathbf{A}[i][j]|$ for $\mathbf{A} \in \mathbb{Z}^{m \times n}$. The proof being a consequence of a result from [Sch86, Theorem 17.2].

Proposition 4.4. *[IS16] Given $\boldsymbol{A} \in \mathbb{Z}^{m \times n}$ and $\boldsymbol{b} \in \mathbb{Z}^m$, for $n \geq 2$, the system $\boldsymbol{A} \cdot \boldsymbol{x} \leq \boldsymbol{b}$ has a solution in $\mathbb{N}^n$ if and only if it has a solution such that $||\boldsymbol{x}|| \leq m^{2n} \cdot ||\boldsymbol{A}||^n \cdot ||\boldsymbol{b}||$.*

The result of this proposition together with the fact that if a cycle in a path schema is iterated a certain number of times then the effect of the iterations can be simulated by some translations allow us to bound the number of times cycles are iterated in order to obtain a run.

Proposition 4.5. *[IS16] Let ρ be a run of S starting from c_0 and an iterated path schema $(P, \boldsymbol{m}) \in ips(\rho)$. There exists a run ρ' starting from c_0 and an iterated path schema $(P, \boldsymbol{m}') \in ips(\rho')$, such that $||\boldsymbol{m}'|| \leq 2^{Poly(\mathtt{size}(S)+\mathtt{size}(c_0))}$, for a polynomial function $Poly(x)$.*

4.6.3 *Upper bounds for the reachability and model-checking problems*

Whereas for flat Translating Counter System, the quantifier free Presbruger formula to characterize the number of cycle iterations in a path schema is of polynomial size in the size of the path schema, it is not anymore the case here and as a consequence it seems difficult to use the result of Proposition 4.5 to guess a value for the vector $\mathbf{m}'$ such that $||\mathbf{m}'|| \leq 2^{Poly(\mathtt{size}(S)+\mathtt{size}(c_0))}$ and then verify it is a solution of the system of equations associated to the path schema without building such a system of equations. In fact, we choose a different path. We use a polynomial-time bounded nondeterministic Turing machine that guesses an iterated path schema and then a NP oracle to check whether a guard has been violated. This last step is possible because we guess the position (up to a certain number of iterations of the last cycle) in the iterated path schema where the guard is violated and we can compute the counter values at this position in polynomial time using matrix exponentiation by squaring. This gives us an $\mathrm{NP}^{\mathrm{NP}}$ algorithm for CS-ControlReach and since for flat Affine Counter Systems CS-Reach can be reduced to CS-ControlReach, we have the following result.

Theorem 4.8. *[IS16]* CS-ControlReach *and* CS-Reach *are in Σ_2^P for disjunction free flat Affine Counter Systems with the finite monoid property .*

Remark. *It is true that this upper bound holds only for disjunction free counter systems but we believe we could extend it to all the class flat Affine Counter Systems with the finite monoid property by adapting the method presented in [DDS15] to eliminate the disjunctions (see Proposition 4.2). This would allow us to match the lower bound from Theorem 4.7. However in [IS16] we did not wish to enter into the heavy details of eliminating disjunctions and our goal was to focus more on the specific aspects of Affine Counter Systems.*

Finally thanks to the stuttering theorem for Past LTL and FO presented in [DDS15] and [DDS13] and by adapting the method used for reachability, we were able to provide upper bounds for the model-checking of these two specification languages. Here again to avoid to track in the unfolding of our systems the atomic guards that are valid, we consider a restriction of these specification languages that do not use atomic guards from $\mathtt{G}(\mathtt{C}_n)$. The specifications are hence built over constrained alphabets of the form $(at, \emptyset, \Gamma)$. This means

that the specifications can only speak about the atomic propositions of the counter systems. Past LTL and FO both enjoy a stuttering property that state that there is no need to iterate a cycle too many times to check whether an ultimately periodic word derived from a path schema satisfies a formula. Actually this bound on the number of iterations to consider is polynomial in the temporal depth of the formulas for Past LTL and exponential in the quantifier height of the formulas for FO (this explains the difference in the complexity for the model-checking problem). This consideration allows us to obtain the following result.

Theorem 4.9. *[IS16] For specifications over constrained alphabets of the form $(at, \emptyset, \Gamma)$ and for disjunction free flat Affine Counter System with the finite monoid property, we have that:*

1. CS-Model-Checking$(Past\ LTL)$ *is in* Σ_2^P.
2. CS-Model-Checking(FO) *is* Pspace*-complete.*

Remark. *For this latter result, we believe as well that we could remove the restrictions on the constrained alphabet and on the guards of the considered systems and obtain the same complexity upper bounds by adapting the results of Proposition 4.2 but this would lead to heavy writing development without bringing any novelty scientifically speaking.*

5

VERIFICATION OF FREEZE LTL

In the previous chapter, we have studied the model-checking problem for some linear time specification languages over counter systems. The presented languages were extensions of classical languages to which we added atomic guards over the counter values as atomic properties, but with such mechanisms it is not possible to compare the values of the counters between configurations. For instance, none of the presented languages allows to say that a counter will have the same value infinitely often (without specifying the value) or that after some time, the value of a counter will change at each step. In order to express such properties, we need a specification language where the values of the counters at different positions can be compared: a good candidate is the logic Freeze LTL, which extends the temporal logic LTL with a storing mechanism to register some data values at some point and compare them later on. I present in this chapter the results obtained in [DS10; BQS17; BQS19], where we have investigated the decidability and the complexity of the model-checking problem of Freeze LTL over some specific classes of counter systems.

The rise of logical formalisms like Freeze LTL comes from the desire to have specification languages for data words, i.e. words where each position is labelled by a letter from a finite alphabet and by a datum from an infinite alphabet. Typical examples are runs of counter systems, timed words accepted by timed automata [AD94] and runs of systems with unboundedly many parallel components (data are component indices) [BB07]. The extension to trees makes also sense to model XML documents with values, see e.g. [Boj+09; JL07]. In order to really speak about data, known logical formalisms for data words/trees contain a mechanism that stores a value and tests it later against other values, see e.g. [Boj+11; DL09]. However, the satisfiability problem for these logics becomes easily undecidable even when stored data can be tested only for equality. For instance, first-order logic for data words restricted to three individual variables is undecidable [Boj+11] and Freeze LTL restricted to a single register is undecidable over infinite data words [DL09]. By contrast, decidable fragments for the satisfiability problems have been found in [Boj+11; DLN07; DL09] either by imposing syntactic restrictions (bound the number of registers, constrain the polarity of temporal formulas, etc.) or by considering subclasses of data words (finiteness for example). Similar phenomena occur with metric temporal logics and timed words [OW07]. A key point for all these logical formalisms is the ability to store a value from an infinite alphabet, which is a feature also present in models of register automata, see e.g. [BPT03; NSV04; Sego6]. However, the storing mechanism has a long tradition (apart from its ubiquity in programming languages) since it appeared for instance in real-time logics [AH89] (the data are time values) and in so-called hybrid logics (the data are node addresses), see an early undecidability result with reference pointers in [Gor96].

During my PhD thesis, I have studied the decidability of the model-checking problem of Freeze LTL and of first-order logic with data equality tests over simple One Counter Systems [DLS10] and whereas many problems are decidable for this class of systems, it turned out that these two model-checking problems are undecidable. However we showed that decidability can be regained by considering deterministic systems (i.e. counter systems where in any configuration there is only a single possible successive configuration).

CONTRIBUTIONS. In [DS10], we have studied the decidability status of the model-checking problem of Freeze LTL over counter systems and we have found some decidable classes by restricting either the considered counter systems or the set of Freeze LTL formulas. In particular we have shown for the flat fragment of Freeze LTL that the model-checking problem can be reduced to the repeated reachability problem for an extension of counter systems where parameterised guards are allowed. Parameterised guards consist in comparisons with a variable, and in that context the verification problem tries to assign values to these variables in order to build some runs. Furthermore this reduction preserves the number of counters. However this translation did not allow us to obtain decidability for the model-checking of a fragment of flat Freeze LTL over simple One Counter Systems, the reason being that we did not at this time know how to solve the repeated reachability problem for simple One Counter Systems with parameterised guards.

Later on, in [Lec+16; Lec+18], Lechner et al show that the repeated reachability problem for One Counter Systems with parameterised guards is decidable. They provided a reduction to the satisfiability problem of an existential Presburger formula, whose size is doubly exponential in the size of the system. This allowed them to obtain a 2NEXPTIME upper bound for the model-checking of flat Freeze LTL over simple One Counter Systems. In [BQS17; BQS19], we have proved that this problem is in fact NEXPTIME-complete using a different technique based on the simulation of a simple One Counter System with parameterised guards by an alternating two-way automaton working over infinite words.

5.1 SPECIFYING COUNTER SYSTEMS WITH FREEZE LTL

In this section, we present a variant of the temporal logic LTL with registers (also known as Freeze LTL) in order to reason about runs from counter systems. When dealing with a single counter as in [DLS10], the datum stored in a register is the current counter value and equality tests are performed between a register value and the current counter value. But when taking into consideration general counter systems, a register can store the value of a counter x and test it later against the value of counter x' with the possibility that $x \neq x'$. Below, we present the syntax and semantics of Freeze LTL in the context of verification of counter systems.

As for counter systems, we consider $\mathrm{AT} = \{p_1, p_2, \ldots\}$ a countably infinite set of propositional variables and a natural $n \geq 1$ (representing the number of counters in the counter systems undez analysis), the formulas of the logic Freeze LTL, denoted $\mathrm{LTL}^{\downarrow}$, are defined by the following grammar:

$$\phi ::= \mathtt{p} \mid \uparrow_r^c \mid \neg\phi \mid \phi \wedge \phi \mid \mathtt{X}\phi \mid \phi \mathtt{U} \phi \mid \phi \mathtt{R} \phi \mid \downarrow_r^c \phi$$

where $\mathtt{p} \in \mathrm{AT}$, $c \in [1, n]$ and $r \in \mathbb{N} \setminus \{0\}$. Intuitively, the modality $\downarrow_r^c$ is used to store the value of the counter x_c into the register r and the atomic formula $\uparrow_r^c$ holds true if the value stored in the register r is equal to the current value of the counter x_c. We say that an occurrence of $\uparrow_r^c$ within the scope of some freeze quantifier $\downarrow_r^c$ is bound by it; otherwise it is free. A sentence is a formula with no free occurrence of any $\uparrow_r^c$.

In our context models of $\mathrm{LTL}^{\downarrow}$ are runs of Affine Counter Systems of dimension n. To define the semantics of $\mathrm{LTL}^{\downarrow}$, we consider hence an Affine Counter System $S = (Q, \mathtt{C}_n, \Delta, \mathbf{1})$ and an infinite run $\rho := (q_0, \mathbf{v}_0) \xrightarrow{\delta_0} \cdots \xrightarrow{\delta_{m-1}} (q_m, \mathbf{v}_m) \xrightarrow{\delta_m} \cdots$. We furthermore need to define a *counter valuation* f which is a finite partial map from $\mathbb{N} \setminus \{0\}$ to $\mathbb{N}$ (we use it to store the

contents of the register). In our semantics, whenever $f(r)$ is undefined for $r \in \mathbb{N} \setminus \{0\}$, then the atomic formula $\uparrow_r^c$ is interpreted as false. Given a position $i \in \mathbb{N}$, the satisfaction relation $\models$ is defined as follows:

$$
\begin{array}{rcl}
\rho, i \models_f \mathtt{p} & \stackrel{\text{def}}{\Leftrightarrow} & \mathtt{p} \in \mathbf{l}(q_i) \\
\rho, i \models_f \uparrow_r^c & \stackrel{\text{def}}{\Leftrightarrow} & r \in dom(f) \text{ and } f(r) = \mathbf{v}_i[c] \\
\rho, i \models_f \neg\phi & \stackrel{\text{def}}{\Leftrightarrow} & \rho, i \not\models_f \phi \\
\rho, i \models_f \phi_1 \wedge \phi_2 & \stackrel{\text{def}}{\Leftrightarrow} & \rho, i \models_f \phi_1 \text{ and } \rho, i \models_f \phi_2 \\
\rho, i \models_f \mathtt{X}\phi & \stackrel{\text{def}}{\Leftrightarrow} & \rho, i+1 \models_f \phi \\
\rho, i \models_f \phi_1 \mathtt{U} \phi_2 & \stackrel{\text{def}}{\Leftrightarrow} & \rho, j \models_f \phi_2 \text{ for some } i \leq j \\
 & & \text{such that } \rho, k \models_f \phi_1 \text{ for all } i \leq k < j \\
\rho, i \models_f \phi_1 \mathtt{R} \phi_2 & \stackrel{\text{def}}{\Leftrightarrow} & \rho, j \models_f \phi_2 \text{ for all } i \leq j \\
 & & \text{or } \rho, j \models_f \phi_1 \text{ for some } i \leq j \text{ and } \rho, k \models_f \phi_2 \text{ for all } i \leq k \leq j \\
\rho, i \models_f \downarrow_r^c \phi & \stackrel{\text{def}}{\Leftrightarrow} & \rho, i \models_{f[r \mapsto \mathbf{v}_i[c]]} \phi
\end{array}
$$

where $f[r \mapsto \mathbf{v}_i(c)]$ denotes the register valuation equal to f except that the register r is mapped to $\mathbf{v}_i[c]$. In the sequel, we omit the subscript "f" in $\models_f$ when sentences are involved.

As for classical linear-time properties, we can define the model-checking problem for LTL$^{\downarrow}$ as follows:

CS-MODEL-CHECKING(LTL$^{\downarrow}$)

Input: An Affine Counter System $S = (Q, \mathtt{C}_n, \Delta, \mathbf{l})$,
an initial configuration $c_0 \in Q \times \mathbb{N}^n$,
and a LTL$^{\downarrow}$ sentence ϕ;

Question: Does there exists an infinite run ρ starting at c_0 such that $\rho, 0 \models \phi$?

Here again we consider the existential version of model checking, this problem can be viewed as a variant of satisfiability in which satisfaction of a formula can be only witnessed within a specific class of data words, namely the runs of a counter system. Note that results for the universal version of model checking will follow easily from those for the existential version when considering fragments closed under negation or deterministic counter machines (for which there will be a unique infinite run).

Example 5.1. *Consider the Translating Counter System depicted in Figure 5.1 where we assume that the set of atomic propositions* AT *is equal to* $\{q_0, q_1, q_2, q_3, q_4\}$ *and the labelling function* $\mathbf{l}$ *is such that* $\mathbf{l}(q_i) = \{q_i\}$ *for all* $i \in [0, 4]$. *We could take as a first LTL*$^{\downarrow}$ *sentence* $\phi_1 = \mathtt{G}(\downarrow_1^1 (q_1 \rightarrow \mathtt{X}(\neg q_1 \mathtt{U}(q_1 \wedge \uparrow_1^2))))$ *(where* $\mathtt{G}\phi$ *is the usual LTL shortcut for* $\neg(\top \mathtt{U} \neg\phi)$*). This formula checks whether there is a run of the counter system verifying that at each instant* t *when the control state* q_1 *is met then it it will be met again in the future and the first time it will be met again the value of the second counter will be the same as the value of the first counter at time* t. *The idea here is to store the value of the first counter at time* t *in the first register and later on when* q_1 *is met again to compare the value of the first register with the value of the second counter. Another interesting LTL*$^{\downarrow}$ *sentence could be the following one:* $\phi_2 = \mathtt{G}(\downarrow_1^1 \downarrow_2^2 (q_4 \rightarrow \mathtt{XG}((\neg \uparrow_1^1) \wedge (\neg \uparrow_2^2))))$. *This latter*

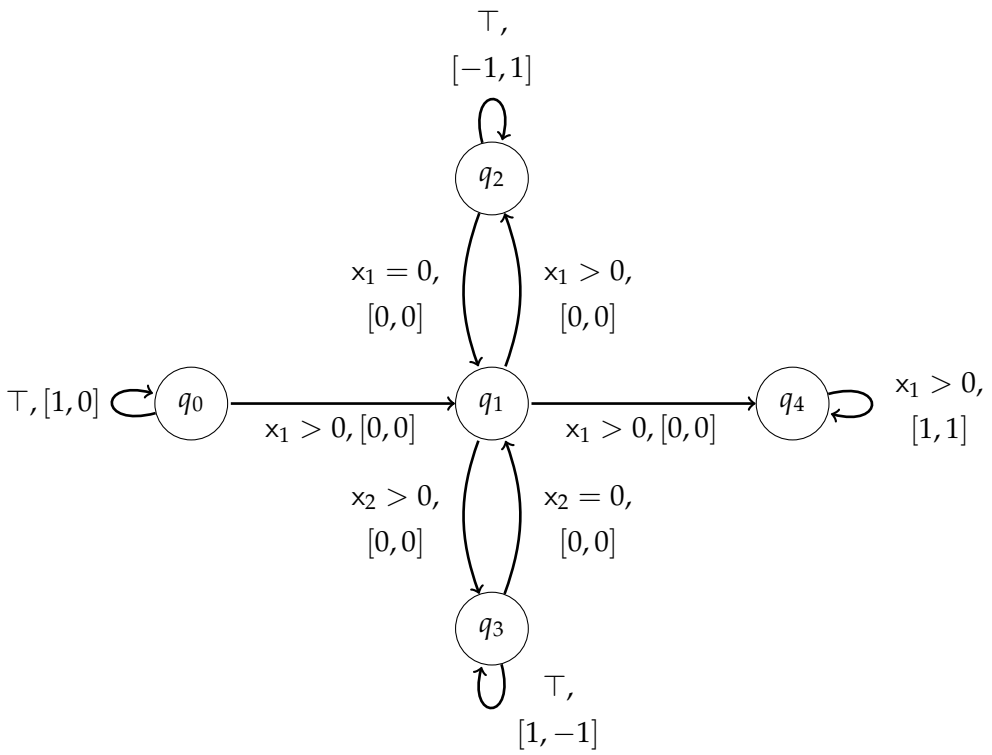

Figure 5.1: A Translating Counter System

formula states that at each instantt when the control state q_4 is met, then both counters always take different values in the sequel of the run than the ones they had at time t . If we take as initial configuration $c_0 = (q_0, \boldsymbol{o})$ for both these LTL$^{\downarrow}$ sentences, there is an infinite run in the Translating Counter System of Figure 5.1 which satisfies it.

5.2 GENERAL RESULTS

5.2.1 *Previously known results*

We began to study the decidability status of CS-Model-Checking(LTL$^{\downarrow}$) over simple One Counter Systems during my PhD and the results we obtained can be found in [DLS10]. Our first result was negative since we showed that we could reduce the halting problem for deterministic Minsky machines into this model-checking problem using carefully the registers of the logic to simulate the values of the counters of the Minsky machine. However, we were able to regain decidability by considering a specific family of simple One Counter Systems, namely deterministic simple One Counter Systems. Before to recall the results we obtained, we provide the formal definition of these systems.

We say that an Affine Counter System $S = (Q, \mathtt{C}_n, \Delta, \mathbf{1})$ is *deterministic* if for all configurations $(q, \mathbf{v}) \in Q \times \mathbb{N}^n$, there is at most one configuration $(q', \mathbf{v}') \in Q \times \mathbb{N}^n$ such that $(q, \mathbf{v}) \to (q', \mathbf{v}')$ (note that in [DS10], the used definition is a bit less restricted since we apply this restriction only to the configurations reachable from the given initial configuration).

In [DLS10], we show that the transition system $\mathfrak{T}(S)$ associated to a deterministic simple One Counter System S has a very specific shape on which it is possible to reason to solve the model-checking problem for LTL$^{\downarrow}$ and we obtained hence the following results.

Theorem 5.1. *[DLS10]* CS-Model-Checking*(LTL$^{\downarrow}$) is undecidable over simple One Counter Systems and is* Pspace*-complete over deterministic simple One Counter Systems (where the updates are encoded in unary).*

As we have already mentioned in the introduction of Chapter 4, in [Dem+10] it is shown that the model-checking problem for an extension of the branching time temporal logic CTL$*$ where the atomic propositions are Presburger definable sets of configurations is decidable for flat Affine Counter Systems with the finite monoid property. This extension, called FOPCTL$*$ allows furthermore the use of quantified first order variables to speak about configurations. The temporal logic LTL$^{\downarrow}$ can hence be viewed as a fragment of this latter logic.

Theorem 5.2. *[Dem+10]* CS-Model-Checking*(LTL$^{\downarrow}$) is decidable over flat Affine Counter Systems with the finite monoid property.*

5.2.2 *Negative results for VASS and Reversal-Bounded Counter Systems*

In [DS10], we have investigated if for some classes of counter systems for which the model-checking problem of LTL is decidable, it is the case as well that CS-Model-Checking(LTL$^{\downarrow}$) is decidable. In particular, we have studied this problem for VASS and Reversal-Bounded Counter Systems. We shall now recall the definitions of these classes of systems.

Reversal-Bounded Counter systems were first introduced in [Iba78] by considering the following restriction: each counter performs only a bounded number of alternations between increasing and decreasing mode. This class of Counter Systems is particularly interesting because it has been shown that each Reversal-bounded Counter System has a semilinear reachability set that can be effectively computed. Furthermore in [FS08], we have extended this class by showing that the same results could be obtained even if one counts only the alternations that occur above a given bound. We provide now a formal definition of such systems.

First we only consider Translating Counter Systems with simple guards, i.e. we assume that each guard is a boolean combination of atomic guards of the form $x_i \sim b$ where $\sim in\{<, \leq, =, >, \geq\}$ and $b \in \mathbb{N}$. In fact, it known that comparing the value of the counters in Reversal-Bounded Counter Systems can break the nice properties of these systems (in [Iba+02] it is proved that for Reversal-Bounded Counter Systems with equality tests between distinct counters the reachability problem is undecidable). To such a Translating Counter System $S = (Q, C_n, \Delta, \mathbf{l})$ and a bound $b \in \mathbb{N}$, we associate an extended transition system $\mathfrak{T}_b(S) = (Q \times \mathbb{N}^n \times \{\texttt{dec}, \texttt{inc}\}^n \times \mathbb{N}^n, \rightarrow_b)$ where $Q \times \mathbb{N}^n \times \{\texttt{dec}, \texttt{inc}\}^n \times \mathbb{N}^n$ corresponds to the set of extended configurations. Intuitively an extended configuration $(q, \mathbf{v}, \mathbf{mode}, \sharp\mathbf{alt})$ records a standard configuration of $\mathfrak{T}(S)$ (with the pair $(q, \mathbf{v})$), **mode** stores the current mode (either decreasing or increasing) for each counter and $\sharp$**alt** stores the number of alternations above b for each counter. The transition relation $\rightarrow_b$ can then be defined as follows. $(q, \mathbf{v}, \mathbf{mode}, \sharp\mathbf{alt}) \rightarrow_b (q', \mathbf{v}', \mathbf{mode}', \sharp\mathbf{alt}')$ if and only if the following conditions hold:

- $(q, \mathbf{v}) \rightarrow (q', \mathbf{v}')$ (in the transition system $\mathfrak{T}(S)$), and,

- for $i \in [1, n]$, the relation described by the following table is verified:

$\mathbf{v}[i] - \mathbf{v}'[i]$	$\mathbf{mode}[i]$	$\mathbf{mode}'[i]$	$\mathbf{v}[i]$	$\sharp\mathbf{alt}'[i]$
> 0	dec	dec	$-$	$\sharp\mathbf{alt}[i]$
> 0	inc	dec	$\leq b$	$\sharp\mathbf{alt}[i]$
> 0	inc	dec	$> b$	$\sharp\mathbf{alt}[i] + 1$
< 0	inc	inc	$-$	$\sharp\mathbf{alt}[i]$
< 0	dec	inc	$\leq b$	$\sharp\mathbf{alt}[i]$
< 0	dec	inc	$> b$	$\sharp\mathbf{alt}[i] + 1$
$= 0$	dec	dec	$-$	$\sharp\mathbf{alt}[i]$
$= 0$	inc	inc	$-$	$\sharp\mathbf{alt}[i]$

As usual, we denote by $\rightarrow_b^*$ the reflexive and transitive closure of $\rightarrow_b$. Following [FSo8], given $b, k \in \mathbb{N}$ we say that a Translating Counter System with simple guards $S = (Q, \mathtt{C}_n, \Delta, \mathbf{l})$ is *k-Reversal-b-Bounded* from an initial configuration $(q_0, \mathbf{v}_0)$ iff for all $(q, \mathbf{v}, \mathbf{mode}, \sharp\mathbf{alt})$ such that $(q_0, \mathbf{v}_0, \mathbf{inc}, \mathbf{0}) \rightarrow_b^* (q, \mathbf{v}, \mathbf{mode}, \sharp\mathbf{alt})$, we have $\sharp\mathbf{alt} \leq k$ (here $\mathbf{inc}$ denotes the vector where all the components are equal to inc). Finally a Translating Counter System with simple guards is said to be *Reversal-Bounded* from $(q_0, \mathbf{v}_0)$ iff it is k-Reversal-b-Bounded from $(q_0, \mathbf{v}_0)$ for some $b, k \in \mathbb{N}$. Note that the Reversal-Bounded Counter Systems introduced by Ibarra in [Iba78] are k-Reversal-0-Bounded Counter Systems in our context. We will call *Ibarra-Reversal-Bounded* Counter Systems such systems.

We have shown in [FSo8] that if S is Reversal-Bounded from $(q_0, \mathbf{v}_0)$, then the set $\{(q, \mathbf{v}) \mid (q_0, \mathbf{v}_0) \rightarrow (q, \mathbf{v})\}$ is definable in Presburger arithmetic (and a corresponding formula can be effectively computed). Consequently we have the following result for Reversal-Bounded Counter System.

Theorem 5.3. *[FSo8]* CS-ControlReach *and* CS-Reach *are decidable for Reversal-Bounded Counter Systems.*

For what concerns the model-checking problem of LTL (which in this chapter can be seen as the restriction of the logic LTL$^{\downarrow}$ without the use of $\uparrow_r^c$ or $\downarrow_r^c$), one classical method,see e.g. [VW86], consists in building from an LTL formula ϕ a Büchi automaton B_ϕ which recognises exactly the sequences of set of atomic propositions satisfying ϕ and then to perform a cross-product with the system under analysis to find an execution satisfying the formula (we recall that this works in our case since we are looking at the existential version of the model-checking problem). It is possible to perform such a cross-product with Counter Systems to obtain a new Counter System for which one wants to check whether there exists an infinite run which visits infinitely often a control-state. Hence if one can solve this latter problem then the model-checking of LTL becomes as well decidable (with a possible blowup in the complexity as the built Büchi automaton can be of exponential size with respect to the LTL formula). Of course, for this translation to work, the cross product has to preserve the restrictions that lead to the decidability of the repeated control-state reachability. We now introduce formally this problem.

CS-REPCONTROLREACH

Input: An Affine Counter System $S = (Q, \mathtt{C}_n, \Delta, \mathbf{l})$,
an initial configuration $(q_0, \mathbf{v}_0) \in Q \times \mathbb{N}^n$
and a set of control states $F \subseteq Q$;

Question: Does there exists an infinite run $\rho := (q_0, \mathbf{v}_0) \xrightarrow{\delta_0} \cdots \xrightarrow{\delta_{m-1}} (q_m, \mathbf{v}_m) \xrightarrow{\delta_m} \cdots$
such that the set $\{i \in \mathbb{N} \mid q_i \in F\}$ is infinite?

In [Jan90], it is shown that CS-REPCONTROLREACH is decidable for VASS. From [Hab97] we know that it is even EXPSPACE-complete and this allows o concluse that the model-checking problem of LTL over VASS is decidable. For what concerns Reversal-Bounded Counter Systems a similar decidability result exists. Indeed CS-REPCONTROLREACH is shown to be decidable for k-Reversal-0-bounded in [Iba+00] and the result has then been extended to Reversal-Bounded Counter Systems in [Sano8]. However, in [DS10] we show that when we extend LTL with the freeze quantifier, the model-checking problem over these two classes of systems becomes undecidable.

To obtain both these negative results, we perform a reduction from the halting problem of deterministic Minsky machines. For VASS, it suffices to remark that one can simulate zero tests (present in the Minsky machines but not in VASS) by storing the initial value of the counters (set to 0) in a register $r = 1$, thanks to $\downarrow_r^1$, and then considering only runs where when we execute in the Minsky machine a zero-test on counter x_i, we take the corresponding transition in the VASS ensuring that the counter is equal to the 0 stored in the register, thanks to $\uparrow_r^i$. For the case of Reversal-Bounded Counter Systems, we use a reduction similar to the one presented in in [Iba+02] in order to prove that in Reversal-Bounded Counter Systems extended with equality tests between distinct counters, the reachability problem of a control state is undecidable. Indeed, assuming that the guards of the form $x = x'$ are allowed, each counter x from the Minsky machine provides two increasing counters x^{inc} and x^{dec}, that counts the number of incrementations on x and the number of decrementations, respectively. Zero-test for x is simulated by a test $x^{inc} = x^{dec}$, that is logically equivalent to $\downarrow_1^{x^{dec}} \uparrow_1^{x^{inc}}$ in LTL$^{\downarrow}$. These two reductions lead to the following result.

Theorem 5.4. *[DS10]* CS-MODEL-CHECKING*(LTL$^{\downarrow}$) over VASS and Reversal-Bounded Counter Systems is undecidable.*

5.2.3 *Regaining decidability*

In order to obtain decidability results for some new cases, a first direction, we have followed in [DS10], was to check whether the results of Theorem 5.4 still hold in the case of deterministic systems. Indeed even if we reduce the halting problem for deterministic Minsky machines, the obtained VASS and Reversal-Bounded Counter Systems we build are intrinsically non deterministic, because the zero-tests are performed by a transition with no test and no update and their validity is ensured by the LTL$^{\downarrow}$ formula. We have in fact shown that this negative result is not true anymore if we focus on deterministic systems.

To obtain this new result, we introduce a new notion on deterministic Affine Counter Systems. We say that a deterministic Affine Counter Systems $S = (Q, \mathtt{C}_n, \Delta, \mathbf{l})$ (with $Q \subseteq \mathbb{N}$)

equipped with an initial configuration $c_0 \in Q \times \mathbb{N}^n$ has the *PA-property* if and only if one can effectively build a Presburger formula $\phi_R(x_0, \ldots, x_{n+1})$ verifying the following assertion: for all $\mathbf{w} \in \mathbb{N}^{n+2}$, we have $\mathbf{w} \models \phi_R$ if and only if there exists a run starting from c_0 of the form $\rho := c_0 \xrightarrow{\delta_0} \cdots \xrightarrow{\delta_{m-1}} c_m \xrightarrow{\delta_m} \cdots$ with $c_{\mathbf{w}[n+2]} = (\mathbf{w}[1], \mathbf{v})$ and $\mathbf{v}[i] = \mathbf{w}[i+1]$ for all $i \in [1, n]$. Since we consider deterministic systems, there is a single run starting from c_0, hence $(\mathbf{w}[1], \mathbf{v})$ is the $\mathbf{w}[n+2]$-th configuration of this run. Consequently the formula $\phi_R(x_0, \ldots, x_{n+1})$ allows to navigate along the run and we can use it to solve the model-checking problem for LTL$^{\downarrow}$ by translating this latter problem into the sastisfiability problem for Presburger arithmetic. The translation is classical for the temporal operators and for what concerns the freeze operator, we use extra variables to simulate the registers where the counter values are stored.

Proposition 5.1. *[DS10]* CS-Model-Checking*(LTL$^{\downarrow}$) is decidable for deterministic Affine Counter Systems with the* PA-property*.*

The positive result of Theorem 5.1 uses partially the fact that deterministic simple One Counter Systems have the PA-property. Among the other deterministic Affine Counter Systems having the PA-property, we have the following classes:

- deterministic flat Affine Counter Systems with the finite monoid property [FL02];
- deterministic Reversal-Bounded Counter Systems [DS10];
- deterministic VASS [DS10].

We get this latter result for deterministic Reversal-Bounded Counter Systems by adding an extra counter to the system which is increased at each step and which allows to monitor the position in the execution and then we use the fact these systems have a semi-linear reachability set. The proof that deterministic VASS have the PA-property is achieved by showing that for these systems the Karp and Miller tree [KM69] is a single path from which we can deduce the Presburger formula characterising the property. As a matter of fact, we come to the following conclusion for deterministic counter systems.

Theorem 5.5. *[DS10]* CS-Model-Checking*(LTL$^{\downarrow}$) over deterministic VASS and deterministic Reversal-Bounded Counter Systems is decidable.*

5.3 Model-checking the flat fragment

5.3.1 *Flat freeze LTL ($\flat$LTL$^{\downarrow}$)*

One of the feature of the logic LTL$^{\downarrow}$ is the ability to use registers which can potentially store an infinite number of different values. For instance, the formula $\mathtt{G}(\downarrow_1^1 \mathtt{XG}(\neg \uparrow_1^1))$ (where $\mathtt{G}\phi$ is the usual LTL shortcut for $\neg(\top \mathtt{U} \neg\phi)$ meaning 'always' ϕ) states that the value taken by the first counter are all distinct and to specify this the formula uses the first register to remember all the different encountered values. We have observed in [DS10], that if one restricts the syntax of LTL$^{\downarrow}$ to limit the number of values that can be stored in registers, then decidability can be restored. We call such restriction the flat fragment of freeze LTL. Formally, this fragment, denoted by $\flat$LTL$^{\downarrow}$, contains a formula ϕ if, for every occurrence of a subformula $\psi = \phi_1 \mathtt{U} \phi_2$ (respectively, $\psi = \phi_2 \mathtt{R} \phi_1$) in ϕ, the following two properties hold:

- If the occurrence of ψ is in the scope of an even number of negations, then the freeze quantifier $\downarrow$ does not occur in ϕ_1;
- If the occurrence of ψ is in the scope of an odd number of negations, then the freeze quantifier $\downarrow$ does not occur in ϕ_2.

Example 5.2. *For instance the formula* $\mathtt{G}(\downarrow^1_1 \mathtt{XG}(\neg \uparrow^1_1))$ *which is equivalent to* $\neg(\top\mathtt{U}(\downarrow^1_1 \mathtt{X}(\top\mathtt{U} \uparrow^1_1)))$ *does not belong to* $\flat LTL\downarrow$ *but its negation is in this fragment. The following formula (where* $\mathtt{F}\phi$ *is a shortcut for* $\top\mathtt{U}\phi$ *meaning 'eventually'* ϕ*) belongs to the flat fragment and it states that at some point the counter 1 takes a value v such that infinitely often counter 2 is equal to v if and only if infinitely often counter 3 is equal to v. and afterwards there is another position from wich the value of counter 4 is always v:* $\mathtt{F} \downarrow^1_1 [(\mathtt{GF} \uparrow^2_1 \Leftrightarrow \mathtt{GF} \uparrow^3_1) \wedge \mathtt{FG} \uparrow^4_1]$.

5.3.2 *Counter systems with parameterised guards*

In order to show that the model-checking of flat Freeze LTL is decidable for some classes of Translating Counter Systems, we use a reduction towards the repeated control-state reachability problem for Translating Counter Systems equipped with parametric variables to which the counters can be compared to and for which we seek an instantiation allowing the system to verify a specification. To perform this reduction, we use the previously discussed fact that formulas of the flat fragment of $\mathrm{LTL}^{\downarrow}$ can only store a finite number of values in their registers and the parametric variables are hence used to store these registered values. We can then build from a flat Freeze LTL formula and a Translating Counter System, a Translating Counter System with parameterised guards with some states to be visited infinitely often by adapting the classical automata based approach to solve the model-checking problem for LTL[VW86].

Given the finite set of counters $\mathtt{C}_n = \{\mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{x}_n\}$ and a finite set of parameters $\Gamma = \{\gamma_1, \ldots, \gamma_k\}$, we call a parameterised guard, a boolean combination of atomic guards of the form $\mathtt{x}_i \sim b$ where $\sim in\{<, \leq, =, >, \geq\}$ and $b \in \mathbb{N} \cup \Gamma$. We denote then by $\mathtt{PG}(\mathtt{C}_n, \Gamma)$ the set of parameterised guards over $\mathtt{C}_n$ and Γ.

Definition 5.1 (Translating Counter System with parameterised guards)**.** *For a natural number* $n \geq 1$*, a Translating Counter System with parameterised guards S of dimension n is a tuple* $(Q, \mathtt{C}_n, \Gamma, \Delta, \mathbf{l})$ *where:*

- Q *is a finite set of control states,*
- Γ *is a finite set of parameters,*
- $\mathbf{l} : Q \to 2^{\mathrm{AT}}$ *is a labeling function, and,*
- $\Delta \subseteq Q \times \mathtt{PG}(\mathtt{C}_n, \Gamma) \times \mathbb{Z}^n \times Q$ *is a* finite *set of edges labelled by guards and affine functions to update the the counter values.*

Note that these systems only differ from Translating Counter Systems by the introduction of parameters in the guards (and the limitation to simple guards). Given a Translating Counter System with parameterised guards $S = (Q, \mathtt{C}_n, \Gamma, \Delta, \mathbf{l})$ and a concretization for the parameters, given by a function $pc : \Gamma \mapsto \mathbb{N}$, we build the Translating Counter System $S_{pc} = (Q, \mathtt{C}_n, \Delta', \mathbf{l})$ from S by replacing in the guards, each parameter $\gamma \in \Gamma$ by the natural $pc(\gamma)$. We can then redefine the repeated control state reachability problem for these systems as follows:

PARAM-CS-CONTROLREACH

Input: A Translating Counter System with parameterised guards $S = (Q, \mathfrak{C}_n, \Gamma, \Delta, \mathbf{l})$, an initial configuration $(q_0, \mathbf{v}_0) \in Q \times \mathbb{N}^n$ and a control state $q_f \in Q$;

Question: Does there exists a concretization $pc : \Gamma \mapsto \mathbb{N}$ and $\mathbf{v} \in \mathbb{N}^n$ such that $(q_0, \mathbf{v}_0) \to^* (q_f, \mathbf{v})$ in $\mathfrak{T}(S_{pc})$?

PARAM-CS-REPCONTROLREACH

Input: A Translating Counter System with parameterised guards $S = (Q, \mathfrak{C}_n, \Gamma, \Delta, \mathbf{l})$, an initial configuration $(q_0, \mathbf{v}_0) \in Q \times \mathbb{N}^n$ and a set of control states $F \subseteq Q$;

Question: Does there exists a concretization $pc : \Gamma \mapsto \mathbb{N}$ and an infinite run $\rho := (q_0, \mathbf{v}_0) \xrightarrow{\delta_0} \cdots \xrightarrow{\delta_{m-1}} (q_m, \mathbf{v}_m) \xrightarrow{\delta_m} \cdots$ in $\mathfrak{T}(S_{pc})$ such that the set $\{i \in \mathbb{N} \mid q_i \in F\}$ is infinite?

We obtain decidability results for these problems for Reversal-Bounded Counter Systems and for One Counter Systems. We say that a Translating Counter System with parameterised guards is (Ibarra)-Reversal-Bounded if the Counter System obtained by replacing all the atomic guards of the form $x_i \sim b$ with b a parameter by $x_i \geq 0$ is (Ibarra)-Reversal-Bounded. This latter class of systems has been studied in [Iba+02] where the authors show that the repeated control state reachability is decidable. To obtain this result, the trick consists in storing the parameter in some extra-counters that are increased at the beginning of the run and to show that in an run it is enough to check each parameterised guard a finite number of times (the atomic guards being convex). This trick allows to reason finally on a classical Ibarra-Reversal-Bounded Counter Systems and apply known results on it.

Theorem 5.6. *[Iba+02]* PARAM-CS-CONTROLREACH *and* PARAM-CS-REPCONTROLREACH *restricted to Ibarra-Reversal-Bounded Counter Systems are decidable.*

The proof to obtain these results can certainly be adapted to Reversal-Bounded Counter Systems, however in [DS10] we did not need this result for the model-checking of the flat fragment of Freeze LTL as we shall explain later on.

5.3.3 *Reduction of the model-checking problem*

As we have already mentioned, given a Translating Counter System S with simple guards, an initial configuration c_0 and a formula ϕ of $\flat\text{LTL}^{\downarrow}$, we can build a Counter System with parameterised guards S' such that there exists an infinite run of S starting at c_0 and satisfying ϕ if and only if there exists an infinite run in S' starting at c_0 which visits infinitely often a set of control states. Moreover this translation from S to S' preserves the number of counter and the operation (apart from the guards with parameters) performed on the counters and if S is (Ibarra)-Reversal-Bounded, so is S'.

To build S', we follow the same path as the automata theoretic approach which allows to build a Büchi automaton from an LTL formula, with the specificity that we do directly

here the cross product with the system S and that we take care of the registers by using parameters. The detail of the construction can be found in [DS10] and we provide here a small example to give an intuition on why this translation is feasible.

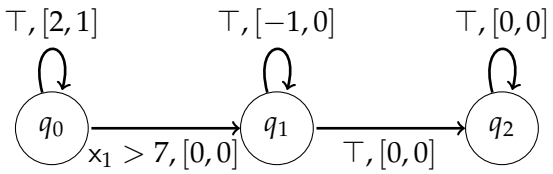

Figure 5.2: A Reversal-Bounded Translating Counter System

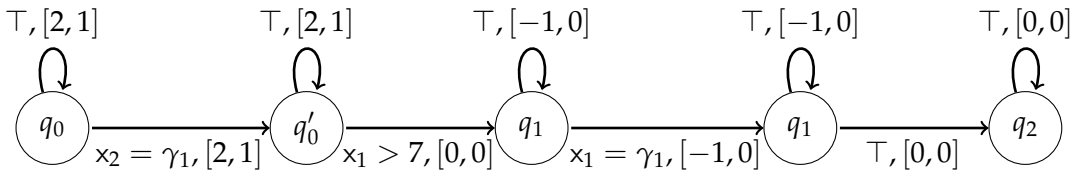

Figure 5.3: A Reversal-Bounded Translating Counter System with parameterised guards

Example 5.3. *The Figure 5.2 provides an example of a Translating Counter System of dimension 2 which is Revarsal-Bounded when equipped with the initial configuration $c_0 = (q_0, \boldsymbol{o})$. We consider then the following $\flat$LTL$^{\downarrow}$ formula $\phi := \mathtt{F}(\downarrow_1^2 \mathtt{XF}(\uparrow_1^1))$. It checks whether there is an execution for which at some point the value of the second counter will eventually (in the strict future) be seen in the first counter. For this, it stores at some point the value of the second counter in the first register and it compares later on the value of the first counter with the content of this register. On Figure 5.3, we provide part of the Translating Counter System with parameterised guards whose role is to check whether the system of Figure 5.2 has a run starting from c_0 satisfying ϕ. We see that the parameter γ_1 plays in a sense the same role as the first register of ϕ, the first time it is encountered it is used to store a value and the next times to test one. The control state q_0' and q_1' are simple copy of q_0 and q_1 used to non-deterministically chose which value of the counter will be stored or tested. If we take as a concretization $pc(\gamma_1) = 4$, we can find in the obtained system an infinite run which visits the control state q_2 infinitely often and deduce that there is a run satisfying ϕ in the first system.*

We obtain then the following result, where CS-Model-Checking($\flat$LTL$^{\downarrow}$) denotes CS-Model-Checking(LTL$^{\downarrow}$) restricted to the flat fragment of LTL$^{\downarrow}$, thanks to this construction.

Proposition 5.2. *[DS10] There is an exponential time translation from* CS-Model-Checking($\flat$LTL$^{\downarrow}$) *restricted to Translating Counter Systems with simple guards to* Param-CS-RepControlReach *and this translation does not change the number of counters, nor the set of non-parameterised operations on these counters and it preserves the (Ibarra)-Reversal-Boundedness of the input systems.*

The exponential blowup comes from the fact that when building the system with parameterised guards, each state is labelled with some subformulas of the provided $\flat$LTL$^{\downarrow}$ formula (the same problem occurs when translating LTL to Büchi automata). Using Theorem 5.6, we directly deduce our first result for the model-checking of $\flat$LTL$^{\downarrow}$ over Translating Counter Systems.

Theorem 5.7. *[DS10] CS-Model-Checking($\flat$LTL$^{\downarrow}$) restricted to Ibarra-Reversal-Bounded Counter Systems is decidable.*

We have furthermore shown that the previous result extends to Reversal-Bounded Counter Systems. One way would have been to adapt the proof of Theorem 5.6, but we have proven that CS-Model-Checking($\flat$LTL$^{\downarrow}$) restricted to Reversal-Bounded Counter Systems reduced to CS-Model-Checking($\flat$LTL$^{\downarrow}$) restricted to Ibarra-Reversal-Bounded Counter Systems. The main idea for this reduction is to encode the counter value in the control state when this value is below the bound b (for a k-Reversal-b-Bounded Counter System). A similar idea was indeed used in [FS08] to show that the reachability set of Reversal-Bounded Counter Systems is semi-linear.

Theorem 5.8. *[DS10] CS-Model-Checking($\flat$LTL$^{\downarrow}$) restricted to Reversal-Bounded Counter Systems is decidable.*

5.3.4 *The specific case of One Counter Systems*

In [DS10], we have as well applied Proposition 5.2 to obtain results for the model-checking of the flat fragment of Freeze LTL over One Counter Systems. However we were not able to show that the model-checking of the whole flat fragment is decidable. The reason being that we relied on existing results presented in [Haa+09], where it is shown that for One Counter Systems with parameterised updates (i.e. in addition to the normal updates, there are some actions that can increment or decrement the counter with parameterized integer constants) the reachability problem of a control state is decidable. It is then easy to use this result in our context by substituting each test of the form $= \gamma_1$ by the following sequence of instructions: decrement by γ_1, perform a zero-test and increment by γ_1. The same technique can be used to encode the tests $\geq \gamma_1$ except that we do not introduce a zero-test between the decrementation (in fact we also add a decrementation by 1 and an incrementation by 1) and the incrementation. However this trick does not allow to deal with parameterised guards of the form $\neq \gamma_1$ or $< \gamma_1$, because the value of the counter cannot be negative. This is the reason why we could only deduce in this first work the decidability of the model-checking of $\flat$LTL$\downarrow$ over One Counter Systems for flat formulas where the operator $\uparrow$ does not occur in the scope of an odd number of negations (we called this fragment the positive flat fragment of LTL$^{\downarrow}$).

Later on, in [Lec+16; Lec+18], the repeated Control State Problem for One Counter Systems with parameterised guards was shown to be decidable by a reduction to the satisfiability problem for quantifier free Presburger arithmetic. Since the built formula has size doubly exponential in the size of the input this allows to obtain a 2NExpTime upper bound for CS-Model-Checking($\flat$LTL$^{\downarrow}$) over One Counter Systems. We will now explain how we improved this result in [BQS17; BQS19].

The first step of our contribution consists in showing how to encode the behavior of One Counter Systems with unary encoding of the constants and parameterised guards into an Alternating Two-Way Automaton. We first recall the definition of this model, the difference with Alternting Büchi Automaton (see Definition 4.2 in Chapter 4) is that here the automaton can navigate on the read word in two directions.

Definition 5.2 (Alternating Two-Way Büchi Automaton). *An Alternating Two-Way Büchi Automaton (A2A) B is a tuple $(Q, \Sigma, \delta, q_0, F)$ where:*

- *Q is a non-empty finite set of states,*
- *Σ is a finite alphabet,*
- *$\delta \subseteq Q \times (\Sigma \cup \{\texttt{first?}\}) \times \mathbb{B}^+(Q \times \{+1, 0, -1\})$ is the transition relation,*
- *$q_0 \in Q$ is the initial state, and,*
- *$F \subseteq Q$ is the set of accepting states.*

A transition $(s, test, \beta) \in \delta$ will also be written $s \xrightarrow{test} \beta$. The symbols $+1$, 0 and -1 are used here to move on the input word and they respectively mean *go left*, *stay* and *go right* whereas `first?` is used to test whether the automaton is at the beginning of the word.

A *run* of B on an ω-word $w = a_0 a_1 a_2 \ldots \in \Sigma^\omega$ is a rooted tree (possibly infinite, but finitely branching) whose vertices are labelled with elements in $Q \times \mathbb{N}$. A node with label (q, n) represents a proof obligation that has to be fulfilled starting from state q and position n in the input word. The root of a run is labelled by $(q_0, 0)$. Moreover, we require that, for every vertex labelled by (q, n) with $k \in \mathbb{N}$ children labelled by $(q_1, n_1), \ldots, (q_k, n_k)$, there is a transition $(s, test, \beta) \in \delta$ such that:

(i) the set $\{(q_1, n_1 - n), \ldots, (q_k, n_k - n)\} \subseteq S \times \{+1, 0, -1\}$ satisfies β,

(ii) $test = \texttt{first?}$ implies $n = 0$, and

(iii) $test \in \Sigma$ implies $a_n = test$.

Note that, similarly to a One Counter System, a transition with move -1 is blocked if $n = 0$, i. e., if B is at the first position of the input word. A run is *accepting* if every infinite branch visits some accepting state from F infinitely often. The language of B is defined as $L(B) = \{w \in \Sigma^\omega \mid \text{there exists an accepting run of } B \text{ on } w\}$. The *non-emptiness problem* for A2As can then be defined as follows:

A2A-NonEmptiness	
Input:	An A2A B;
Question:	Do we have $L(B) \neq \emptyset$?

And here is a result providing an upper bound to solve this latter problem.

Theorem 5.9. *[Sero6]* A2A-NonEmptiness *is in* Pspace.

We shall now see how these automata can help us to solve Param-CS-ControlReach and Param-CS-RepControlReach for One Counter Systems with parameterised guards. Let $S = (Q, \mathtt{C}_n, \Gamma, \Delta, \mathbf{1})$ be a One Counter System with parameterised guards where the updates are encoded in unary. The main idea proposed in [BQS17; BQS19] is to encode a concretization $pc : \Gamma \mapsto \mathbb{N}$ as a word over the alphabet $\Sigma = Gamma \cup \{\square\}$, where $\square$ is a fresh symbol. A word $w = a_0 a_1 a_2 \cdots \in \Sigma^\omega$, with $a_i \in \Sigma$, is called a *parameter word* (over Γ) if $a_0 = \square$ and, for all $\gamma \in \Gamma$, there is exactly one position $i \in \mathbb{N}$ such that $a_i = \gamma$. Said differently, w starts with $\square$ and every parameter occurs exactly once. Then, w determines a concretization $pc_w : \Gamma \to \mathbb{N}$ as follows: if $\gamma = a_i$, then $pc_w(\gamma) = |a_1 \cdots a_{i-1}|_\square$ where

$|a_1 \cdots a_{i-1}|_\square$ denotes the number of occurrences of $\square$ in $a_1 \cdots a_{i-1}$ (note that we start at the second position of w). For example, given $\Gamma = \{\gamma_1, \gamma_2, \gamma_3\}$, both $w = \square\gamma_2\square\square\gamma_1\gamma_3\square^\omega$ and $w' = \square\gamma_2\square\square\gamma_3\gamma_1\square^\omega$ are parameter words with $pc_w = pc_{w'} = \{\gamma_1 \mapsto 2, \gamma_2 \mapsto 0, \gamma_3 \mapsto 2\}$. Note that, for every concretization pc, there is at least one parameter word w such that $pc_w = pc$. From S, an initial configuration $(q_0, \mathbf{v}_0)$ and a subset of states $F \in Q$, we can build an A2A that accepts the set of parameter words w such that there exists an execution in $\mathfrak{T}(S_{pc_w})$ visiting infinitely often some states of F. The simulation proceeds as follows. When the One Counter System increments its counter, the A2A moves to the right to the next occurrence of $\square$. To simulate a decrement, it moves to the left until it encounters the previous $\square$. To mimic the zero test, it verifies that it is currently on the first position of the word. Moreover, it will make use of the letters in Γ to simulate parameterised guards. At the beginning of an execution, the A2A spawns independent copies that check whether the input word is a valid parameter word and during the execution, each time the system encounters a guard, the A2A generates a new branch in charge of verifying that the guard is satisfied.

This construction allows us to reduce in polynomial time Param-CS-ControlReach and Param-CS-RepControlReach for One Counter Systems to A2A-NonEmptiness and as a matter of fact to obtain a Pspace upper bound for these two problems. Actually we show that this upper bound can be improved. For this matter, we use the two following arguments given a One Counter System with unary encoding of the constants and parameterised guards S, an initial configuration $(q_0, \mathbf{v}_0)$ and a final state q_f:

1. the reduction towards A2A allows us to state that if there exists a concretization $pc : \Gamma \mapsto \mathbb{N}$ and $\mathbf{v} \in \mathbb{N}^n$ such that $(q_0, \mathbf{v}_0) \to^* (q_f, \mathbf{v})$ in $\mathfrak{T}(S_{pc})$ then the value of the parameters given by the concretization pc can be bounded by a number exponential in the size of S as well as the values of the counter in the execution from $(q_0, \mathbf{v}_0)$ to $(q_f, \mathbf{v})$;

2. in [Gal76], it is shown that for a One Counter System given two configurations c and c' (where the values of the counters are encoded in binary), one can decide in polynomial time, whether there exists an execution between these two configurations whose counter values is strictly between the values of c and c'.

Thanks to these two results, we are able to propose an NP algorithm to solve Param-CS-ControlReach which works as follows: it first guesses a value for the parameters in Γ (and store it in a binary encoding), then it guesses some configurations of the run where the counter takes value of these parameters (it guesses at most $|Q| \cdot (|\Gamma| + 2)$ such configurations, because there is no need to go twice through the same configuration and the $+2$ is here to deal with the initial and final counter values) and finally it uses the result of [Gal76] to verify in polynomial time whether the proposed execution is feasible. Note that this algorithm allows to deal as well with constants in guards encoded in binary (we treat them as parameter but instead of guessing their value, we impose them).

Furthermore, we remark that to visit a set of states infinitely often, either an execution sees infinitely often the same configuration or the counter values always increase, this allows us to reduce in polynomial time Param-CS-RepControlReach to Param-CS-ControlReach for One Counter Systems with parameterised guards. Finally the lower bound for these two problems is obtained from the fact that the non-emptiness problem for nondeterministic two-way automata over finite words over a *unary* alphabet and two end-markers is NP-

complete [Gal76]. This allows us to obtain the following results for simple One Counter Systems (i.e. where the constants in the updates are encoded in unary)

Theorem 5.10. *[BQS17; BQS19]*
PARAM-CS-REPCONTROLREACH *and* PARAM-CS-CONTROLREACH *restricted to One Counter Systems with updates constants encoded in unary and parameterised guards are* NP*-complete.*

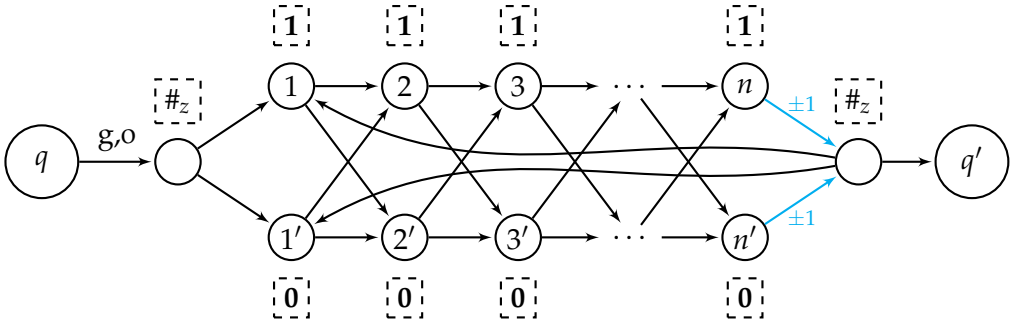

Figure 5.4: The gadget for simulating an transitions (q, g, z, q') with binary update z.

Thanks to Proposition 5.2, we can hence deduce that the model-checking of the flat fragment of freeze LTL restricted to One Counter Systems is in NEXPTIME when the updates constants are encoded in unary. However, we show that the power of LTL allows us to deal with constants in binary by encoding succinct updates in small LTL formulas. In fact, from a simple One Counter System S and a formula ϕ of $\flat$LTL$^{\downarrow}$ we can build another system S' which uses only updates of the form $+1$ or -1 and we can translate the formula ϕ into another formula ϕ' of $\flat$LTL$^{\downarrow}$ whose in charge to both check that there exists an execution of S satisfying ϕ and to select executions of S' which correspond to executions of ϕ by checking that the updates of S are performed correctly in S'. The fundamental key is that there is no exponential blowup while building the formula ϕ'. To do this translation, we encode each transition of S into a some small looping circuit that will be traveled in a repeated way to simulate the updates. Figure 5.4 provides an example of this translation where new added propositions are depicted in dashed boxes; states $1, \dots, n, 1', \dots, n'$, where $n = bits(z)$, represent the bits needed to encode z in binary and at the transitions originating from n and n', the counter is updated by $+1$ or -1, depending on whether z is positive or negative, respectively. The LTL formula ensures then that the circuit is taken a number of times corresponding to the update. This construction gives us an upper bound for the model-checking of $\flat$LTL$^{\downarrow}$ over simple One Counter Systems and the matching lower bound can be deduced from the complexity of model-checking of LTL over One Counter Systems with parameterised and succinct updates [Göl+10].

Theorem 5.11. *[BQS17; BQS19]* CS-MODEL-CHECKING($\flat$*LTL*$^{\downarrow}$*) restricted to One Counter Systems is* NEXPTIME*-complete.*

5.4 SUMMARY OF THE RESULTS

The Table 5.1 sums up the results we have obtained for the model-checking of LTL with the freeze quantifier over Affine Counter Systems. For many cases, we only established the decidability status and did not mention any complexity bound. Some of them can be derived by a careful analysis on the existing studies over the considered models.

	No restriction	Deterministic	Flat formulas
$\mathcal{VASS}$	Undecidable [Thm 5.4]	Decidable [Thm 5.5]	Undecidable [Thm 5.4]
flat $\mathcal{ACS}$ with finite monoid	Decidable [Thm 5.2]	Decidable [Thm 5.2]	Decidable [Thm 5.2]
Reversal-Bounded	Undecidable [Thm 5.4]	Decidable [Thm 5.5]	Decidable [Thm 5.8]
$\mathcal{OCS}$	Undecidable [Thm 5.1]	PSPACE-complete (unary encoding) [Thm 5.1]	NEXPTIME-complete [Thm 5.11]

Table 5.1: Complexity/Decidability of the model-checking problem of $\text{LTL}^{\downarrow}$ over counter systems

6

VERIFICATION OF BRANCHING TIME PROPERTIES

After having presented results on the model-checking of linear time properties in the previous chapters, we present now the works we have done concerning the model-checking of branching time properties over counter systems. As the considered formalisms allow, as linear time properties, to express the reachability of a control state, we know that the model-checking of such properties is undecidable even for simple counter systems equipped with two counters (as a consequence of Theorem 3.1). However as for the model-checking of linear time properties, decidability can be regained by using restrictions on the considered models.

Whereas linear properties consider the different executions individually, branching time properties allow to reason on the tree of executions of a system and can for instance express the possibility of a system to be in a state from which two differents successor states can be reached. As for linear time properties, many specification languages have been proposed to reason on such trees among which the Computation Tree Logic (CTL), its extension CTL*, which allows to express as well linear time properties like fairness, and the modal μ-calculus, a fixpoint logic with some modal operators to consider one successor or all the successors of a configuration. For what concerns the algorithmic point of view, when considering finite state systems, solving the model-checking problem is not harder and sometimes easier when going from linear time properties to branching time properties. For instance, as we have seen previously the model-checking of LTL over Kripke Structures is PSPACE-complete [SC85], but the model-checking of CTL over these systems turns to be in PTIME, the complexity for CTL* is as well PSPACE-complete and for what concerns the μ-calculus, there is an algorithm in NP $\cap$ co-NP.

When taking as models infinite state systems and more specifically counter systems, it turns out that dealing with branching time formalisms tends to be more complex than having linear time specifications. For instance for VASS, the model-checking of the linear time temporal logic LTL is EXPSPACE-complete but as soon as one uses a branching reasoning which can express the existence of a successor, the model-checking becomes undecidable. One can in fact easily simulates the zero tests of deterministic Minsky machines with such a mechanism. In fact, as said in [EN94], "no natural and useful branching time temporal logics for Petri nets seems to be decidable". We shall see here that there are some ways to overcome this undecidability result for VASS.

For one counter systems, it turns out that model-checking branching time properties is feasible. It has in fact be shown that the model-checking of CTL properties and of the modal μ-calculus over one counter systems is PSPACE-complete when the updates are encoded in unary [GL13; Ser06] and is EXPSPACE-complete with a binary encoding [Göl+10; Ser06]. The results for the modal μ-calculus is obtained by solving two-player games played on the graph of a one counter system where the winning condition is expressed as a parity condition. Indeed one can translate the model-checking problem for modal μ-calculus into this kind of games (see e.g. [GTW02]). We will see that we exploit as well this translation in the case of VASS.

In this chapter we focus on developing model-checking algorithms for branching time temporal logics over VASS and translating counter systems.

CONTRIBUTIONS. After having described the precise complexity of the model-checking of linear time properties over flat Translating Counter Systems (see Chapter 4), in [DDS14; DDS18] we have determined the complexity of the model-checking of the branching time temporal logic CTL* (where arithmetical constraints can be used as atomic propositions). The decidability of this problem is given in [Dem+10], however the proof is based on a translation towards an exponential size formula of Presburger arithmetic and we showed that this blow-up can be avoided.

We have then studied the model-checking of a restricted fragment of the modal μ-calculus over VASS. As already mentioned, in its full generality when considering VASS, the model-checking of branching time logics is undecidable, but we succeed in finding an unstudied fragment for which we can design an algorithm. This work was originally motivated by the verification a probabilistic version of VASS where some transitions can be taken with a certain probability. In [AHM07], the authors have studied verification of infinite-state Markov chains, and they have shown that if the models respect a property which they call decisiveness, then verification of certain qualitative and quantitative probabilistic properties can be achieved. In particular they have shown that Probabilistic VASS, which extend VASS by adding a probabilistic distribution to each set of transitions available in a configuration, induce decisive Markov chains. In Probabilistic VASS, all choices are hence probabilistic and we wanted to study whether some properties could be verified if we considered both probabilistic and non-deterministic choices in VASS obtaining hence infinite-state Markov Decision Processes. It appears that for finite Markov Decision Processes, the verification of qualitative properties (which consists in determining whether a properties hold almost surely or not) can be encoded into verification problems of μ-calculus formulas (interpreted over the underlying system of the Markov Decision Process) as shown in [Cha+09]. However such a translation cannot be trivially adapted to any infinite state Markov Decision Process. Following this line of study, in [Abd+13], we showed that the model-checking of a fragment of the modal μ-calculus is decidable for VASS, by relying on a classical translation towards two player games played on the transition system associated to a VASS and in [Abd+16a] we exploited this decidability result to show that we can verify some qualitative properties on Markov Decision Processes induced by VASS.

6.1 MODEL-CHECKING OF CTL* OVER FLAT TRANSLATING COUNTER SYSTEMS

In this section, we will present the results we obtained in [DDS14; DDS18] concerning the model-checking of flat Translating Counter Systems with the branching time logic CTL*.

6.1.1 *Specifying counter systems with CTL**

As for the linear time temporel logic Past LTL presented in Chapter 4, the branching time logic we consider can both speak about the atomic propositions labelling each state of a counter system but as well use some atomic guards to state specification on the counter values. As for counter systems, we consider $\mathtt{C} = \{x_1, x_2, \ldots\}$ an infinite set of counters (variables interpreted over non-negative integers), $\mathtt{C}_n = \{x_1, x_2, \ldots, x_n\}$ its restriction to n

counters and $AT = \{p_1, p_2, \ldots\}$ a countably infinite set of propositional variables. We recall that we denote by $G(C_n)$ atomic guards of the form $\sum_{i=1}^{n} a_i \cdot x_i \sim b$ where the a_i's are in $\mathbb{Z}$ and $b \in \mathbb{N}$.

The formulas of the branching time temporal logic CTL* are then given by the following grammar:

$$\phi ::= p \mid g \mid \neg\phi \mid \phi \wedge \phi \mid X\phi \mid \phi U \phi \mid E\phi$$

where p belongs to at and g belongs to $G(C_n)$. Our version of CTL* is defined as the standard one, see e.g. [EH86], except that the Kripke structures are replaced by transitions systems of counter systems and at the atomic level, arithmetical constraints are allowed. To define the semantics of this logic, we consider hence an Affine Counter System $S = (Q, C_n, \Delta, \mathbf{l})$. We need as well to introduce new notations on runs. We recall that a run ρ starting from c_0 in S is an infinite path in the associated transition system $\mathfrak{T}(S)$ denoted as:

$$\rho := c_0 \xrightarrow{\delta_0} \cdots \xrightarrow{\delta_{m-1}} c_m \xrightarrow{\delta_m} \cdots$$

where $c_i \in Q \times \mathbb{N}^n$ and $\delta_i \in \Delta$ for all $i \in \mathbb{N}$ and a finite run is defined similarly by considering finite paths in $\mathfrak{T}(S)$. We will say that a run is *maximal* if it is either infinite or finite and its last configuration c_m has no successor configuration (i.e. there does not exist a configuration c such that $c_m \to c$). For such a run and $i \geq 0$, we let $\rho(i) = c_i$. Given a finite run $\rho := c_0 \xrightarrow{\delta_0} \cdots \xrightarrow{\delta_{m-1}} c_m \xrightarrow{\delta_m} c_{m+1}$, we write $|\rho| = m + 1$ its length and if ρ is an infinite run, we set $|\rho| = \omega$ (for all $n \in \mathbb{N}$, we have $n \leq \omega$). We are now ready to provide the semantics of CTL*. In this case, the satisfaction relation $\models$ is defined as follows, where ϕ is a CTL* formula, ρ is a maximal run of S and $i \in \mathbb{N}$ is a position in the run such that $i < |\rho|$:

$$\begin{aligned}
\rho, i \models p &\overset{\text{def}}{\Leftrightarrow} \rho(i) = (q, \mathbf{v}) \text{ and } p \in \mathbf{l}(q) \\
\rho, i \models g &\overset{\text{def}}{\Leftrightarrow} \rho(i) = (q, \mathbf{v}) \text{ and } \mathbf{v} \models g \\
\rho, i \models \neg\phi &\overset{\text{def}}{\Leftrightarrow} \rho, i \not\models \phi \\
\rho, i \models \phi_1 \wedge \phi_2 &\overset{\text{def}}{\Leftrightarrow} \rho, i \models \phi_1 \text{ and } \rho, i \models \phi_2 \\
\rho, i \models X\phi &\overset{\text{def}}{\Leftrightarrow} i + 1 \leq |\rho| \text{ and } \rho, i+1 \models_f \phi \\
\rho, i \models \phi_1 U \phi_2 &\overset{\text{def}}{\Leftrightarrow} \rho, j \models \phi_2 \text{ for some } i \leq j \leq |\rho| \\
&\qquad \text{such that } \rho, k \models \phi_1 \text{ for all } i \leq k < j \\
\rho, i \models E\phi &\overset{\text{def}}{\Leftrightarrow} \text{there is a maximal run } \rho' \text{ such that} \\
&\qquad \rho'(0) = \rho(i) \text{ and } \rho', 0 \models \phi
\end{aligned}$$

As usual, we use the shortcuts $F\phi$ for the formula $\top U \phi$ (meaning 'eventually ϕ') and $G\phi$ for $\neg F \neg \phi$ (meaning 'always ϕ') and $A\phi$ for the formula $\neg E \neg \phi$. Given a CTL* formula ϕ, an Affine Counter System S and a configuration c of S, we write $S, c \models \phi$ if and only if there exists a maximal run ρ starting from c and such that $\rho, 0 \models \phi$.

As linear-time properties, we can define the model-checking problem for CTL* as follows:

CS-MODEL-CHECKING(CTL*)	
Input:	An Affine Counter System $S = (Q, \mathtt{C}_n, \Delta, \mathbf{l})$, an initial configuration $c_0 \in Q \times \mathbb{N}^n$, and a CTL* formula ϕ;
Question:	Do we have $S, c_0 \models \phi$?

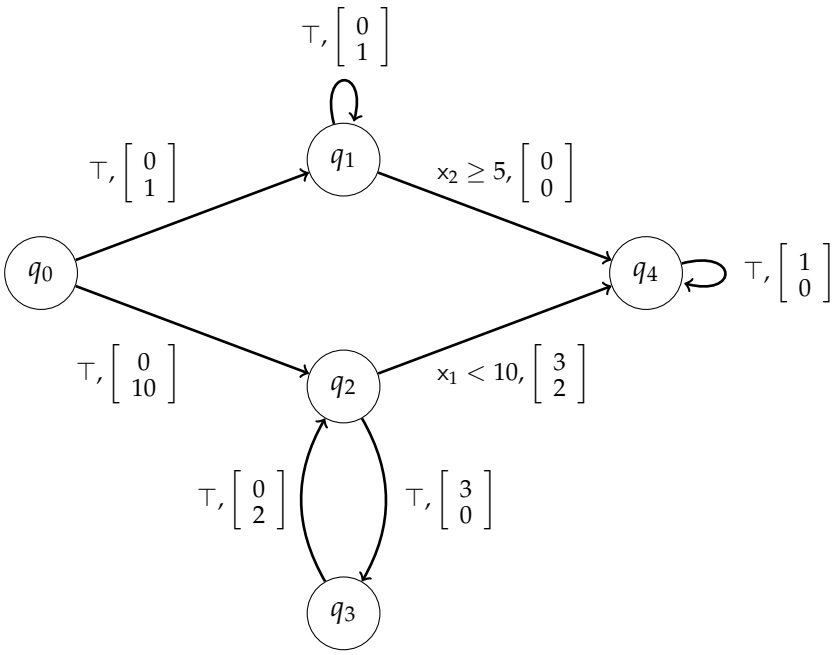

Figure 6.1: A flat Translating Counter System

Example 6.1. *We consider the flat Translating Counter System S of dimension 2 depicted in Figure 6.1 equipped with the initial configuration $c_0 = (q_0, \boldsymbol{o})$ (and where all the states are labelled with the empty set). First, if we take a look at the CTL* formula $\phi := \mathtt{AG}(\mathtt{EF}(x_1 \geq x_2))$ which specifies, that in all the runs, from each encountered configuration, there is a run reaching a configuration where the value of the first counter is bigger than the one of the second counter, then we have $S, c_0 \models \phi$. On the other hand, if we take the CTL* formula $\phi' := \mathtt{AF}(x_2 > 15)$, then $S, c_0 \not\models \phi'$ because there is a run starting at c_0 which goes to q_2 and then loops on q_4 for which the value of the second counter is always strictly smaller than 15.*

It is well known that the branching time temporal logic CTL* allows to express strictly more properties than the linear time temporal logic LTL and that if we restrict CTL* such that every modal operator (`X` or `U`) must be directly preceded by a path quantifier *one*, then we get the branching time temporal logic CTL which is incomparable with LTL (as for instance you can not express fairness properties in CTL where you need formula of the shape $\mathtt{GF}\phi$), as it is explained in [BK08].

6.1.2 *Reduction to the satisfiability problem for Presburger arithmetic*

Because of the undecidability of the reachability of a control state in translating counter machines of dimension 2, it is clear that CS-MODEL-CHECKING(CTL*) is undecidable. However, we know from [Dem+10] that if we restrict ourselves to flat Translating Counter Systems then decidability can be regained (actually the result still holds for flat Affine Counter Systems with the finite monoid property). The proof of this later result is based on a reduction towards the satisfiability problem for Presburger arithmetic. It exploits the fact that in a flat Counter System, every run can be represented by a minimal path schema of the form $p_1(l_1)^* \cdots p_{k-1}(l_{k-1})^* p_k(l_k)^\omega$, where each p_i is a finite path and each l_i is a loop, together with an arithmetical constraint characterising how many times each loop is taken. It is the same technique we used for the model-checking of linear time properties (see Section 4.4 of Chapter 4). In order to perform the model-checking of CTL*, given a run represented by such an iterated path schema, it is easy to encode formulas of the form $\mathtt{p}$, $\mathtt{g}$, $\mathtt{X}\phi$ and $\phi \mathtt{U} \phi$ into a formula of the Presburger arithmetic. To deal with the path quantifier, i.e. formulas of the form $\mathtt{E}\phi$, the translation relies on the fact that the number of such minimal path schemas is finite but exponential, hence it is possible to enumerate the runs thanks to a disjunction. The main issue is that the cost of this translation, due to the enumeration of the path schemas, is exponential in the size of the counter system.

In [DDS14; DDS18], we improved this translation by encoding minimal path schemas and their iterations into some vectors. The trick consists in attributing a number to each transition and each single loop in the system and then we can check thanks to a formula of the Presburger arithmetic whether a vector corresponds to the description of a minimal path schema. Consequently, we can avoid the exponential blowup when building the formula to encode the model-checking of CTL*, by replacing the concrete enumeration of all minimal path schemas by an existential guess of vectors encoding minimal path schemas. This allows to obtain the following result.

Theorem 6.1. *[DDS14; DDS18] There is a logarithmic-space reduction from from* CS-MODEL-CHECKING*(CTL*) restricted to flat Translating Counter Systems to the satisfiability problem for Presburger arithmetic.*

6.1.3 *Lower bound*

We have as well shown in [DDS14; DDS18] that it is possible to reduce the satisfiability problem for Presburger arithmetic into an instance of CS-MODEL-CHECKING(CTL*) restricted to flat Translating Counter Systems. We shall explain now briefly this reduction. We consider a Presburger arithmetic formula $\psi := Q_1 z_1 Q_2 z_2 \dots Q_n z_n \phi'(z_1, \dots, z_n)$ in prenex normal form, i.e. with $Q_1, Q_2, \dots, Q_n \in \{\exists, \forall\}$ and ϕ' is a quantifier-free formula. From this formula, we build the flat Translating Counter System S_ϕ depicted in Figure 6.2 (where the transitions without label carry the guard $\top$ and the update $\mathbf{0}$).

We then build a formula ψ in CTL* whose atomic formulas are among $q_1, \dots, q_{n+1}$ (also abusively understood as control states) such that $S_\phi, (q_0, \mathbf{0}) \models \psi$ if and only if ϕ is satisfiable in Presburger arithmetic. Intuitively, each variable z_i from ϕ is taken care of by the ith loop (that can only increment the ith counter). Additionally, the quantifications from ϕ are simulated in the formula ψ by using $\mathtt{EF}$ or $\mathtt{AG}$, depending whether the first-

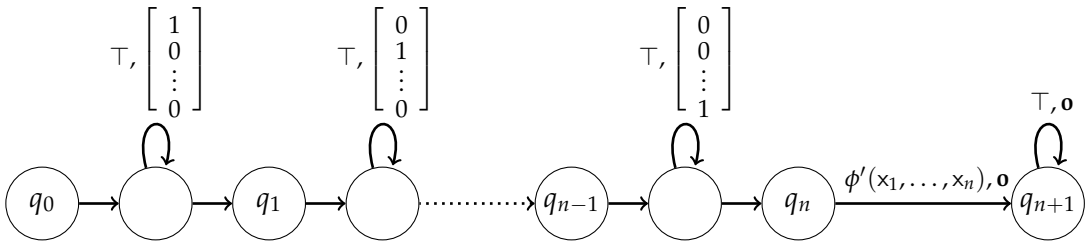

Figure 6.2: The flat Translating Counter System S_ϕ

order quantification is existential or universal. Formally, we introduce the formulas ψ_i with $i \in [1, n+1]$ as follows:

$$\psi_i := \begin{cases} \mathtt{EF}(q_i \wedge \psi_{i+1}) & i \leq n \text{ and } Q_i = \exists \\ \mathtt{AG}(\neg q_i \vee \psi_{i+1}) & i \leq n \text{ and } Q_i = \forall \\ \mathtt{EF} q_{n+1} & i = n+1 \end{cases}$$

The formula ψ corresponds finally to ψ_1. Note that this formula only uses modalities of the form EF and AG. This construction leads to the following result.

Theorem 6.2. *[DDS14; DDS18] There is a logarithmic-space reduction from the satisfiability problem for Presburger arithmetic to* CS-Model-Checking*(CTL*) restricted to flat Translating Counter Systems.*

6.2 Model-checking of μ-calculus over VASS

In this section, we present the results we obtained concerning the model-checking problem of the positive μ-calculus over VASS [Abd+13]. We shall see that to solve it, we use a classical reduction towards two-player games with parity conditions and we show how to solve such games played on the arena generated by a VASS under some specific conditions (since the general case is undecidable).

Remark. *As we shall see we do not take into account atomic propositions in the rest of this chapter. Consequently in order to use simpler notations, when dealing with Affine Counter System (and their various restrictions), we will from now on omit the labelling function which will be of no use.*

6.2.1 *Specifying counter systems with the positive μ-calculus*

In order to simplify the presentation, the version of the μ-calculus we present here uses the control states of the counter systems as atomic propositions. This logic uses least and greatest fixpoints on the set of configurations of a counter system and allows two 'temporal' modalities $\Box$ and $\Diamond$ to speak respectively of all successors of a set of configurations or one successor. We consider an Affine Counter System $S = (Q, \mathtt{C}_n, \Delta)$. The syntax of the positive μ-calculus L_μ^{pos} is given by the following grammar:

$$\phi ::= q \mid X \mid \phi \vee \phi \mid \phi \wedge \phi \mid \Diamond\phi \mid \Box\phi \mid \mu X.\phi \mid \nu X.\phi$$

where $q \in Q$ and X belongs to a countable set of variables $\mathcal{X}$.

Free and bound occurences of variables are defined as usual and a formula is said to be close if it has no free variable. Note that we do not use any negation in our syntax but negation can be pushed inward by the usual dualities of fixpoints and the negation of an atomic proposition referring to a control state can be expressed by a disjunction of propositions referring to all the the other control states.

We now give the interpretation of a formula of L_μ^{pos} over the Affine Counter System $S = (Q, \mathtt{C}_n, \Delta)$. For this matter, we define an environment $\rho : \mathcal{X} \mapsto 2^{Q \times \mathbb{N}^n}$ which associates to each variable a subset of configurations. Given such an environment ρ, a formula ϕ in L_μ^{pos} characterises a subset of configurations denoted by $[\![\phi]\!]_\rho$ and defined inductively as follows:

$$\begin{array}{lcl}
[\![q]\!]_\rho & = & \{c \in Q \times \mathbb{N}^n \mid c = (q, \mathbf{v}) \text{ for some } \mathbf{v} \in \mathbb{N}^n\} \\
[\![X]\!]_\rho & = & \rho(X) \\
[\![\phi_1 \vee \phi_2]\!]_\rho & = & [\![\phi_1]\!]_\rho \cup [\![\phi_2]\!]_\rho \\
[\![\phi_1 \wedge \phi_2]\!]_\rho & = & [\![\phi_1]\!]_\rho \cap [\![\phi_2]\!]_\rho \\
[\![\Diamond\phi]\!]_\rho & = & \{c \in Q \times \mathbb{N}^n \mid \exists c' \in [\![\phi]\!]_\rho \text{ s.t. } c \to c'\} \\
[\![\Box\phi]\!]_\rho & = & \{c \in Q \times \mathbb{N}^n \mid \forall c' \in Q \times \mathbb{N}^n . c \to c' \text{ implies } c' \in [\![\phi]\!]_\rho\} \\
[\![\mu X.\phi]\!]_\rho & = & \bigcap\{C \subseteq Q \times \mathbb{N}^n \mid [\![\phi]\!]_{\rho[X \leftarrow C]} \subseteq C\} \\
[\![\nu X.\phi]\!]_\rho & = & \bigcup\{C \subseteq Q \times \mathbb{N}^n \mid C \subseteq [\![\phi]\!]_{\rho[X \leftarrow C]}\}
\end{array}$$

where the notation $\rho[X \leftarrow C]$ is used to define an environment equal to ρ on every variable except on X where it returns C. We recall that $(2^{Q \times \mathbb{N}^n}, \subseteq)$ is a complete lattice and that for every $\phi \in L_\mu^{pos}$ and every environment ρ, the function $G : 2^{Q \times \mathbb{N}^n} \mapsto 2^{Q \times \mathbb{N}^n}$, which associates to $C \subseteq Q \times \mathbb{N}^n$ the subset $G(C) = [\![\phi]\!]_{\rho[X \leftarrow C]}$, is monotonic. Hence by the Knaster-Tarski Theorem the set $[\![\mu X.\phi]\!]_\rho$, respectively $[\![\nu X.\phi]\!]_\rho$, is the least fixpoint, respectively the greatest fixpoint of G and it is well defined. Finally, we denote by $[\![\phi]\!]$ the subset of configurations $[\![\phi]\!]_{\rho_0}$ where ρ_0 is the environment assigning the empty set to each variable.

We define then the model-checking problem for the positive μ-calculus in our context as follows:

CS-MODEL-CHECKING(L_μ^{pos})

Input: An Affine Counter System $S = (Q, \mathtt{C}_n, \Delta)$, an initial configuration $c_0 \in Q \times \mathbb{N}^n$, and a close formula $\phi \in L_\mu^{pos}$;

Question: Do we have $c_0 \in [\![\phi]\!]$?

As we have said, we can reduce most of the model-checking problems we have considered so far over counter systems into a model-checking problem for μ-calculus formulas (if we do note allow atomic propositions over the values of the counters) as this latter logic is more expressive (except for the logic freeze LTL). Indeed there exists for instance translation from LTL to L_μ^{pos} and as well from CTL* to L_μ^{pos} (see e.g. [GTW02]). However in our context, these translations would not have been very useful, as we do not know for instance whether CS-MODEL-CHECKING(L_μ^{pos}) is decidable over flat Translating Counter Systems and even if it is the case, using such a translation might not have given the nice complexity bounds

we obtained in the previous sections, as it might have a cost in terms of complexity (the translation of an LTL formula into L_μ^{pos} generates for instance an exponential blowup).

Of course, in its full generality, CS-Model-Checking(L_μ^{pos}) is undecidable (since as for the other temporal logics, it can express control state reachability problem). This negative result still holds even if we consider as models VASS (in fact the model-checkling of CTL is already undecidable for VASS) but surprisingly we shall see in the rest of this chapter that we are able to obtain decidability results for a nice fragment of L_μ^{pos}, which will allow us to establish results for the verification of Markov Decision Processes induced by VASS.

Example 6.2. *If we consider an Affine Counter System $S = (Q, \mathtt{C}_n, \Delta)$ with a specific state $bad \in Q$, then one can show that the L_μ^{pos} formula $\phi := \nu X. \bigvee_{q \in Q \setminus \{bad\}} q \wedge \Box X$ characterizes the configurations that cannot reach configurations of the shape $(bad, \boldsymbol{v})$ with $\boldsymbol{v} \in \mathbb{N}^n$. Formally, we have in fact $[\![\nu X. \bigvee_{q \in Q \setminus \{bad\}} q \wedge \Box X]\!] = \{c \in Q \times \mathbb{N}^n \mid \nexists \boldsymbol{v} \in \mathbb{N}^n \text{ s.t. } c \to^* (bad, \boldsymbol{v})\}$.*

6.2.2 *A detour via two-player games*

We shall now present two-player games played on the transition system of a VASS and we shall see how in some cases, we can solve such games and how this allows us to solve CS-Model-Checking(L_μ^{pos}) over VASS under some specific conditions. Note that for the general definitions, we could have introduced games played on the general transition systems of an Affine Counter Systems but in [Abd+13] we have studied only such games for VASS.

Definition 6.1 (Parity VASS game)**.** *A* Parity VASS game *G is a tuple $(Q, Q_1, Q_2, \mathtt{C}_n, \Delta, col)$ is a tuple such that :*

- *$(Q, \mathtt{C}_n, \Delta)$ is VASS,*
- *(Q_1, Q_2) is a partition of the set of states Q, and,*
- *$col : Q \mapsto \mathbb{N}$ is a coloring function.*

The transition system associated to the VASS games will define an arena for a turn based two players (Player 1 and Player 2) game. From a configuration $(q, \mathbf{v})$ with $q \in Q_1$ [resp. $q \in Q_2$], Player 1 [resp. Player 2] will play and propose a successor configuration. The goal of Player 1 from an initial configuration is to build an infinite execution such that the highest color that appears infinitely often will be even, this is what is commonly called the parity condition, and this no matters what are the choices of Player 2. We shall now formalize this aspect.

Remark. *First, we consider only deadlock-free games this means that for each configuration $c \in Q \times \mathbb{N}^n$ there always exists at least one successor configuration $c' \in Q \times \mathbb{N}^n$ such that $c \to c'$. This restriction is here mostly to simplify some aspects and we could deal as well with deadlocks but in that case we only need to be careful on defining well what it means for Player 1 to win in case a deadlock is reached. Classcically, the player which is blocked, i.e. when the state of the deadlock-configuration belongs to him, loses.*

We consider a Parity VASS game $G = (Q, Q_1, Q_2, \mathtt{C}_n, \Delta, col)$. The definition of configurations, finite and infinite runs are the same ones as for the underlying VASS. For $id \in \{1, 2\}$, a strategy for Player id is a mapping that assigns to each finite run $\rho := c_0 \xrightarrow{\delta_0} \cdots \xrightarrow{\delta_{m-1}} c_m$ with $c_m \in Q_{id} \times \mathbb{N}^n$ a configuration c such that $c_m \to c$ (since G is deadlock-free such a

configuration always exists). We denote by Σ_{id} the set of strategies of Player id. We say that an infinite run $\rho := c_0 \xrightarrow{\delta_0} \cdots \xrightarrow{\delta_{m-1}} c_m \cdots$ respects a strategy π of Player id if and only if for each $i \in \mathbb{N}$ if $c_i \in Q_{id} \times \mathbb{N}^n$ then $\pi(c_0 \xrightarrow{\delta_0} \cdots \xrightarrow{\delta_{i-1}} c_i) = c_{i+1}$. Given a configuration $c \in Q \times \mathbb{N}^n$, a strategy π_1 for Player 1 and a strategy π_2 for Player 2, there exists an unique run starting from c, which respects both π_1 and π_2, and we shall denote it with $\texttt{run}(c, \pi_1, \pi_2)$.

To an infinite run $\rho := (q_0, \mathbf{v}_0) \xrightarrow{\delta_0} \cdots \xrightarrow{\delta_{m-1}} (q_m, \mathbf{v}_m) \xrightarrow{\delta_m} \cdots$, we associate the set of colors it sees infinitely often defined by $\texttt{InfCol}(\rho) = \{d \in \mathbb{N} \mid \{i \in \mathbb{N} \mid col(q_i) = d\} \text{ is infinite } \}$. This allows us to define a parity winning condition for Player 1: we say that a run ρ is a winning run for Player 1 if and only if $\max(\texttt{InfCol}(\rho))$ is even, otherwise it is winning for Player 2 (note that since the number of states of a VASS is finite, such a maximum necessarily exists). Finally, we say that c_0 is a winning configuration for Player 1 if and only if there exists a strategy π_1 for Player 1 such that for all strategies π_2 of Player 2, $\texttt{run}(c_0, \pi_1, \pi_2)$ is a winning run for Player 1.

Thanks to these previous definitions, we can define the parity game problem for VASS as follows:

PARITY-VASS-GAME	
Input:	A Parity VASS game G, and an initial configuration c_0;
Question:	Is c_0 a winning configuration for Player 1?

In the sequel, we shall denote by $\texttt{Win}(G)$ the set of winning configurations for Player 1 in the parity VASS game G.

Parity games in the case of finite state systems have been intensively studied. It was first shown that solving such games is in NP∩co-NP [EJS93], this result can be obtained by showing that it is enough to consider memoryless strategies, i.e. strategies whose choice for the next state only depends on the current state. This result was then refined in [Jur98] where it is proven that determining whether a state is winning for Player 1 is in UP∩co-UP. It is still an open problem to know whether this problem can be solved in polynomial time and many works have been done to propose new approaches (see e.g. [Cze+19a] where the authors propose a quasi-polynomial time algorithm framework).

When such games are played on an arena defined by an infinite state system, such as Parity VASS games, determining whether a configuration is winning for Player 1 can be harder to solve and in fact PARITY-VASS-GAME is an undecidable problem (we shall explain in the sequel why). However there are still some specific cases where decidability can be regained as for instance when the games is played on the transition systems of a one counter system [Ser06]. Figure 6.3 shows how to reduce the halting problem for deterministic Minsky machines into a Parity VASS game. On this figure, we have drawn the encoding of the instructions (1) $L : \mathtt{x}_1 := \mathtt{x}_1 + 1$; `goto` L' and (2) $J :$ `if` $\mathtt{x}_2 = 0$ `then goto` J' `else` $\mathtt{x}_2 := \mathtt{x}_2 - 1$; `goto` J''. On this Figure, we assume that circle states belong to Player 1 and the square states to Player 2; all the states carry the color 1 except the halting state L_F (not depicted on the Figure) which belongs to Player 1, carries the color 0 and is equipped with a self-loop not modifying the counter value. Consequently, the only way for Player 1 to win the game is to reach this state L_F. The game fully simulates the Minsky machine and the

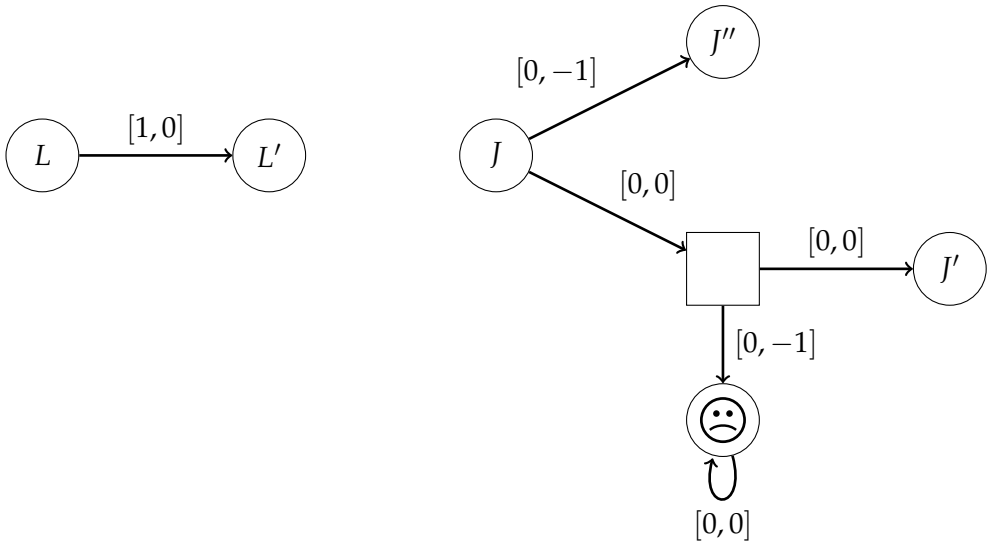

Figure 6.3: Encoding the instructions of a Minsky machine in Parity VASS game

trick relies in the encoding of the zero-test: to achieve this we let Player 1 the option to cheat by choosing the path leading from J to J' for any counter value, but if it does so when the counter is not equal to 0 then Player 2 can punish him by bringing the game in the state ☹ where Player 1 loses.

Example 6.3. *Figure 6.4 provides an example of a Parity VASS games. Here again states of Player 1 are represented by circles and squares are used to characterize states of Player 2. All states are colors with* 1 *except* q_4 *which carries the color* 2*. The configuration* $(q_0, \boldsymbol{o})$ *is winning for Player 1 but the strategies to follow is a bit involved. In fact, to win, Player 1 has to visit infinitely often state* q_4*, for this he needs to have a second counter value bigger than* 10*. He has the ability to increase as much as he wants this counter in state* q_0 *but then the first counter is increased as well from the same amount. Furthermore, if the game reaches state* q_2 *with a strictly positive value on the first counter, then Player 1 loses because Player 2 will have the ability to loop forever between* q_2 *and* q_3*. To avoid this trap, Player 1 can use the loop on the control state* q_1 *in order to decrease until* 0 *the value of the first counter (when in* q_2 *the first counter is equal to* 0*, Player 2 cannot move to* q_3*). To sum up, a winning strategy for Player 1 consists in taking 10 times the loop on* q_0*, then moving to state* q_1 *and taking 10 times the self loop, then moving to the configuration* $(q_2, [0, 10])$ *from which Player 2 can only move to* $(q_4, [0, 10])$ *and finally Player 1 can come back to* $(q_0, \boldsymbol{o})$ *and repeat the whole scenario.*

Remark. *As explained in [Abd+13], Parity VASS games are closely related to Energy Parity Games [CRR12]. Both these games manipulate integer variables, the difference is that in the latter case, if a transition makes one of the counter goes to a negative value, it can be taken but in that case Player 2 wins the game. In [CRR12; Cha+10], the authors has proven that the* unknown initial credit problem *for Energy Parity Games is* coNP*-complete. This problem consists in determining given a control state* q_0 *whether there exists a value for the counters* $\boldsymbol{v}_0$ *such that* $(q_0, \boldsymbol{v}_0)$ *is winning for Player 1. We shall see that we rely on these results. In [BJK10], the authors have shown that without any Parity condition they can decide whether a configuration is winning in Energy Games. However in its full generality deciding whether a configuration is winning in Energy Parity Games was an open problem and people did not know as well how to compute the sets of winning configurations. We answered these two questions as we shall see in the next subsection.*

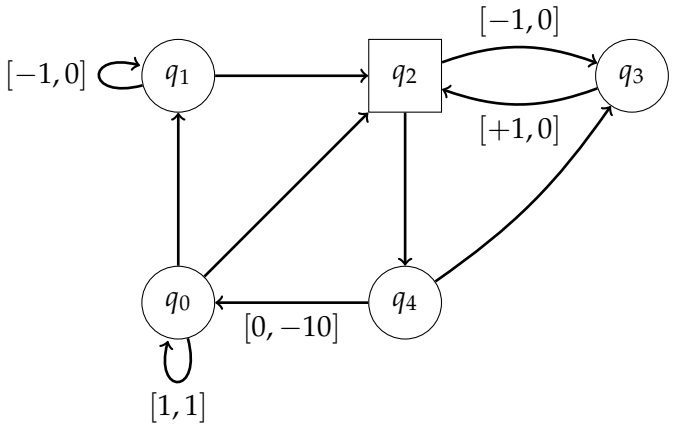

Figure 6.4: A Parity VASS game

We now explain the reason why we present these games. One way to solve the μ-calculus model-checking problem for a given Kripke structure is to encode the problem into a parity game [GTW02]. The idea is to construct a parity game whose states are pairs, where the first component is a state of the structure and the second component is a subformula of the give μ-calculus formula. States of the form $(q, \Box\phi)$ and $(q, \phi \wedge \psi)$ belong to Player 2 and the other states belong to Player 1. The colors are then assigned to reflect the nesting of least and greatest fixpoints. In [Abd+13], we have shown that this construction can be adapted to solve the model-checking problem for L_μ^{pos} over VASS.

Proposition 6.1. *[Abd+13] Let S be a VASS, c_0 an initial configuration of S and ϕ a close formula of L_μ^{pos}. One can construct in polynomial time a Parity VASS Game $G(S, \phi)$ equipped with an initial configuration c_0' such that c_0' is a winning configuration for Player 1 in $G(S, \phi)$ if and only if $c_0' \in [\![\phi]\!]$.*

Since both PARITY-VASS-GAME and CS-MODEL-CHECKING(L_μ^{pos}) are undecidable, at the moment this reduction is not very helpful but we will shall see now how to regain decidability.

6.2.3 *Regaining decidability*

One aspect that leads to undecidability is the ability of Player 2 in the games to decrease the counter, in fact if we do not allow the transitions going out of a state of Player 2 to modify any of the counters valuess then the reduction from the halting problem for deterministic Minsky machines presented previously cannot be done anymore. Indeed we have proven that this restriction allows to regain decidability.

We will say that a Parity VASS game $G = (Q, Q_1, Q_2, \mathtt{C}_n, \Delta, col)$ is *single-sided* if and only if for all transitions $(q, true, \mathbf{b}, q')$ with $q \in Q_2$, we have $\mathbf{b} = \mathbf{0}$. For instance, the Parity VASS game depicted in Figure 6.4 is not single-sided because of the transition between q_2 and q_3. In [RSB05], the authors shows that one could solve single-sided VASS game with coverability objectives (the objective for Player 1 boils down to reach a specific state). In [Abd+13], we have proven that PARITY-VASS-GAME restricted to single-sided Parity VASS game is decidable. We shall now give the main ideas to obtain this result.

We consider a single-sided Parity VASS game $G = (Q, Q_1, Q_2, \mathtt{C}_n, \Delta, col)$. In [Abd+13], we propose an algorithm to compute $\mathtt{Win}(G)$. Since this set is potentially infinite, we first need to show why it can be represented finitely. We define the order $\preceq \subseteq (Q \times \mathbb{N}^n) \times (Q \times \mathbb{N}^n)$ over configurations of G as follows: $(q, \mathbf{v}) \preceq (q, \mathbf{v}')$ if, and only if, $q = q'$ and $\mathbf{v} \leq \mathbf{v}'$. This order $\preceq$ is a well-quasi-order and we show that $\mathtt{Win}(G)$ is upward-closed for this order. As a matter of fact, this latter set has a finite number of minimal elements that can be used to represent it. In order to compute these minimal elements, we reason on abstract configurations. A partical configuration $\gamma = (q, \mathbf{v})$ is a pair in $Q \times (\mathbb{N} \cup \{*\})^n$. The intuition behind such a partial configuration is that the $*$ corresponds to any counter values that can be instantiated. We then define the domain of partial configuration $dom(\gamma) = \{i \in [1, n] \mid \mathbf{v}(i) \neq *\}$ and its concretization as $[\![\gamma]\!] = \{(q, \mathbf{v}') \in Q \times \mathbb{N}^n \mid \mathbf{v}'(i) = \mathbf{v}(i) \text{ for all } i \in dom(\gamma)\}$. We also extend the order $\preceq$ to partial configurations as follows: given two partial configurations $\gamma = (q, \mathbf{v})$ and $\gamma' = (q, \mathbf{v}')$, we have $\gamma \preceq \gamma'$ if and only if $q = q'$ and $dom(\gamma) = dom(\gamma')$ and $\mathbf{v}[i] \leq \mathbf{v}'[i]$ for all $i \in dom(\gamma)$. Here again $\preceq$ over $Q \times (\mathbb{N} \cup \{*\})^n$ is a well-quasi-order. Then to compute $\mathtt{Win}(G)$, we rely on the *Valk-Jantzen* lemma [VJ85] which can be stated as follows in our terminology.

Lemma 6.1. *[VJ85] Let $U \subseteq Q \times \mathbb{N}^n$ be an upward-closed set. Then the minimal elements of U can be computed if and only if we can decide whether $[\![\gamma]\!] \cap U \neq \emptyset$ for any partical configuration γ.*

We have shown in [Abd+13], that we can encode the Energy Parity games from [CRR12; Cha+10] in single-sided Parity VASS games and vice-versa. We use then the fact that for Energy Parity games, the unknown initial credit problem is decidable as a basic block for our procedure to solve Parity-VASS-Game restricted to single-sided parity VASS game. In our terms, the unknown initial credit problem can be expressed as follows: given a partial configuration γ with $dom(\gamma) = \emptyset$, deciding whether $[\![\gamma]\!] \cap \mathtt{Win}(G)$ is empty or not is a co-NP-complete problem [CRR12; Cha+10].

In order to compute $\mathtt{Win}(G)$, in [Abd+13] we reason by induction on the size of the domain of abstract configurations $k \in [0, n]$ and we show, adapting Lemma 6.1 that we can compute the minimal elements of the set $W_k = \{\gamma \in Q \times (\mathbb{N} \cup \{*\})^n \mid |dom(\gamma)| = k \text{ and } \mathtt{Win}(G) \cap [\![\gamma]\!] \neq \emptyset\}$. We have just explained how to deal with the case $k = 0$. To show knowing W_{k-1} that we can compute W_k we rely on an adaptation of the classical Karp-Miller forward analysis algorithm [KM69] to build another Parity VASS game where we only need to check if there is a winning configuration winning from an initial given control state (which we can do similarly to the way we treated the case $k = 0$).

Theorem 6.3. *[Abd+13]* Parity-VASS-Game *restricted to single-sided Parity VASS games is decidable.*

This result was then later refined in [Col+17], where the authors show that Parity-VASS-Game restricted to single-sided Parity VASS games is a 2ExpTime-complete problem (the lower bound was established in [CS14]).

Remark. *In [RSB05], the authors state that they cannot decide whether Player 1 can win a single-sided VASS game where the winning conditions is given by a LTL formula, which seems as contradiction with the previous theorem. In fact usually LTL objectives can be translated into a Parity condition. This result was shown in a Master thesis and it seems that the main difference with what we show here is due to who is winning in case the game reaches a deadlock configuration. In our case,*

we assume that if we have deadlock , then the player whose state is in the deadlock configuration loses, and this is why we can get rid of deadlock by encoding this condition adding extra states. But if one assumes that if a deadlock is met, then Player 1 wins, no matter what is the current configuration, then we cannot get rid of deadlock (keeping the single-sided condition) and we cannot decide whether Player 1 can win the game under parity condition.

The result of the previous theorem together with Proposition 6.1 allows us to obtain as well a decidability result for model-checking of the positive μ-calculus over VASS. First, we reuse the notion of single-sided over VASS by saying that a VASS $S = (Q, \mathtt{C}_n, \Delta)$ is (Q_1, Q_2)-*single-sided* (shortly *single-sided*) if and only if the subset of states Q_1 and Q_2 forms a partition of Q such that for all transitions $(q, true, \mathbf{b}, q')$ with $q \in Q_2$, we have $\mathbf{b} = \mathbf{0}$. The guarded fragment L_μ^{sv} of L_μ^{pos} for single-sided VASS is then defined by guarding the $\Box$ operator with a predicate that enforces the control states to be in Q_2. Formally, the syntax of L_μ^{sv} is given by the following grammar:

$$\phi ::= q \mid X \mid \phi \vee \phi \mid \phi \wedge \phi \mid \Diamond\phi \mid Q_2 \wedge \Box\phi \mid \mu X.\phi \mid \nu X.\phi$$

where Q_2 stands for the formula $\bigvee_{q \in Q_2} q$. The semantics of these formulas interpreted over a single-sided VASS is then the same as for L_μ^{pos} and we can define an adequate model-checking problem.

CS-MODEL-CHECKING(L_μ^{sv})

Input: A single-sided VASS $S = (Q, \mathtt{C}_n, \Delta)$, an initial configuration $c_0 \in Q \times \mathbb{N}^n$, and a close formula $\phi \in L_\mu^{sv}$;

Question: Do we have $c_0 \in [\![\phi]\!]$?

In [Abd+13], we have shown that the Parity VASS game $G(S, \phi)$ from Proposition 6.1 built from a single-sided VASS S and a formula ϕ in L_μ^{sv} is equivalent to a single-sided VASS game. By combining the results of this latter proposition and of Theorem 6.3, we directly obtained the next result.

Theorem 6.4. *[Abd+13]* CS-MODEL-CHECKING(L_μ^{sv}) *is decidable.*

Furthermore our proof techniques allows us to state that given a single-sided VASS $S = (Q, \mathtt{C}_n, \Delta)$ and a close formula $\phi \in L_\mu^{sv}$ then the set of configurations $[\![\phi]\!]$ is upward-closed (with respect to the order $\preceq$ we introduced previously) and we can effectively compute its finite set of minimal elements.

6.3 QUALITATIVE VERIFICATION OF MARKOV DECISION PROCESSES INDUCED BY VASS

We shall now see how the result of Theorem 6.4 allowed us in [Abd+16a] to deduce some results for the qualitative verification of Markov Decision Processes induced by VASS.

6.3.1 *Markov Decision Processes*

We first recall the definition of Markov Decision Processes which correspond to systems where one can find both non-determinism and probabilistic choices (they extend in a certain sense Markov Chains which do not have non-determinism but only a probabilistic transition relation).

Definition 6.2 (MDP). *A Markov Decision Process (MDP) M is a tuple $(\Gamma, \Gamma_1, \Gamma_P, \rightarrow, prob)$ where:*

- Γ *is a countable set of configurations,*
- (Γ_1, Γ_P) *is a partition of* Γ*,*
- $\rightarrow \subseteq \Gamma_1 \times \Gamma$ *is the non-deterministic transition relation, and,*
- $prob : \Gamma_P \rightarrow \texttt{Dist}(\Gamma_1)$ *is a partial function corresponding to the probabilistic transition relation.*

Γ_1 are called the configurations of Player 1 and Γ_P are the probabilistic configurations.

Remark. *Usually, in MDPs, the non-deterministic transition relation comes with an action alphabet, however for the verification problem we address, such an alphabet is not useful and we choose to represent MDPs as games played between a non-deterministic player (Player 1) and a probabilistic player (Player P).*

Given two configurations γ, γ' in Γ, we will sometimes use the notation $\gamma \rightarrow \gamma'$ instead of $(\gamma, \gamma') \in \rightarrow$. We will say that a configuration $\gamma \in \Gamma_1$ is a deadlock if there does not exist $\gamma' \in \Gamma$ such that $\gamma \rightarrow \gamma'$. We denote by Γ_1^{df} the set of configurations of Player 1 which are not a deadlock (df stands here for deadlock free). Similarly a configuration $\gamma \in \Gamma_P$ is a deadlock if $prob(\gamma)$ is not defined. A *finite play* of the MDP M is a finite sequence of configurations $\gamma_0\gamma_1 \dots \gamma_k$ such that for all $i \in [0, k-1]$, if $\gamma_i \in \Gamma_1$ then $\gamma_i \rightarrow \gamma_{i+1}$ and otherwise $prob(\gamma_i)(\gamma_{i+1}) > 0$. We say that such a play starts from the configuration γ_0. An *infinite play* is an infinite sequence $\rho \in \Gamma^\omega$ such that any of its prefix is a finite play. A play is said to be *maximal* it is infinite or it is finite and it ends in a deadlock configuration. These latter plays are called *deadlock plays*. We denote by Ω the set of maximal plays.

A *scheduler* in the MDP $M = (\Gamma, \Gamma_1, \Gamma_P, \rightarrow, prob)$ is a function $\pi : \Gamma^* \cdot \Gamma_1^{df} \mapsto \Gamma$ that assigns to a finite sequence of configurations ending with a configuration in Γ_1^{df} a successor configuration such that for all $\rho \in \Gamma^*$, $\gamma \in \Gamma_1^{df}$ and $\gamma' \in \Gamma$, if $\pi(\rho \cdot \gamma) = \gamma'$ then $\gamma \rightarrow \gamma'$. We denote by Π the set of schedulers. Given a scheduler $\pi \in \Pi$, we say that a finite play $\gamma_0\gamma_1 \dots \gamma_k$ respects the scheduler π if for all $i \in [0, k-1]$, we have that if $\gamma_i \in \Gamma_1$ then $\pi(\gamma_0 \dots \gamma_i) = \gamma_{i+1}$. Similarly we say that an infinite play ρ respects the scheduler π if every finite prefix of ρ respects π. Given an initial configuration γ and a scheduler π, we denote by $\texttt{Plays}(M, \gamma, \pi)$ the set of all maximal plays of M that start from γ and respect π.

Once a starting configuration γ and a scheduler π have been chosen, the MDP is reduced to an ordinary stochastic process. Given a measurable set of plays $\mathcal{A} \subseteq \Omega$, we denote by $\mathbb{P}(M, \gamma, \pi, \mathcal{A})$ the probability of event $\mathcal{A}$ for the plays in $\texttt{Plays}(M, \gamma, \pi)$. The notation $\mathbb{P}^{sup}(M, \gamma, \mathcal{A})$ will then be used to represent the maximal probability of the event $\mathcal{A}$ starting from γ and is defined as follows: $\mathbb{P}^{sup}(M, \gamma, \mathcal{A}) = \sup_{\pi \in \Pi} \mathbb{P}(M, \gamma, \pi, \mathcal{A})$. Similarly $\mathbb{P}^{inf}(M, \gamma, \mathcal{A}) = \inf_{\pi \in \Pi} \mathbb{P}(M, \gamma, \pi, \mathcal{A})$.

6.3.2 *VASS-MDP and their associated verification problems*

A first extension of VASS with probabilities have been proposed in [AHM07], however the authors proposed a model where only probabilistic choices were allowed. In [Abd+16a], we extend this latter model with non-deterministic choices made by a controller (or equivalently Player 1).

Definition 6.3 (VASS-MDP). *A VASS-MDP S is a tuple $(Q, Q_1, Q_P, \mathtt{C}_n, \Delta, \tau)$ such that:*

- *$(Q, \mathtt{C}_n, \Delta)$ is VASS such that for each $(q, true, \boldsymbol{b}, q') \in \Delta$ with $q \in Q_P$, we have $q' \in Q_1$,*
- *(Q_1, Q_P) is a partition of the set of states Q, and,*
- *$\tau : \Delta \mapsto \mathbb{N} \setminus \{0\}$ is a function assigning to each transition a weight.*

As for MDP, Q_1 corresponds to the states of Player 1, whereas Q_P are the states of the probabilistic player. A VASS-MDP $S = (Q, Q_1, Q_P, \mathtt{C}_n, \Delta, \tau)$ induces a MDP $M_S = (\Gamma, \Gamma_1, \Gamma_P, \rightarrow, prob)$ where:

- $\Gamma = Q \times \mathbb{N}^n$, $\Gamma_1 = Q_1 \times \mathbb{N}^n$ and $\Gamma_P = Q_P \times \mathbb{N}^n$,
- for all $\gamma \in \Gamma_1$ and $\gamma' \in \Gamma$, we have $\gamma \rightarrow \gamma'$ if and only if $\gamma \dashrightarrow \gamma'$ (assuming $(\Gamma, \dashrightarrow)$ is the transition system associated to the VASS $(Q, \mathtt{C}_n, \Delta)$),
- for all $\gamma \in \Gamma_P$, if there exists γ'' such that $\gamma \dashrightarrow \gamma''$ (i.e. γ is not a deadlock configuration) then for $\gamma' \in \Gamma_1$ we have $prob(\gamma)(\gamma') = \tau(\delta)/(\Sigma_{\{\delta' \in \Delta \mid \exists \gamma''. \gamma \overset{\delta'}{\dashrightarrow} \gamma''\}} \tau(\delta'))$ if $\gamma \overset{\delta}{\dashrightarrow} \gamma'$ and $prob(\gamma)(\gamma') = 0$ otherwise. If there does noet exists γ'' such that $\gamma \dashrightarrow \gamma''$, then $prob(\gamma)$ is not defined.

Remark. *The definition of the MDP M_S is well founded. Indeed, when defining $prob(\gamma)(\gamma')$ in the case $\gamma \dashrightarrow \gamma'$, we are sure that $\{\delta' \in \Delta \mid \exists \gamma''. \gamma \overset{\delta'}{\dashrightarrow} \gamma''\}$ is not empty and $\Sigma_{\{\delta' \in \Delta \mid \exists \gamma''. \gamma \overset{\delta'}{\dashrightarrow} \gamma''\}} \tau(\delta')$ is never equal to 0. Also we could have restricted the weight function τ to be assigned only to transitions δ such that $source(\delta) \in Q_P$ since we do not take into account the weights assigned to the other transitions.*

We say that a VASS-MDP $S = (Q, Q_1, Q_P, \mathtt{C}_n, \Delta, \tau)$ is deadlock-free if and only if in the transition system $(\Gamma, \dashrightarrow)$ associated to the VASS $(Q, \mathtt{C}_n, \Delta)$, for all configurations $\gamma \in \Gamma$, there exists a configuration $\gamma' \in \Gamma$ such $\gamma \dashrightarrow \gamma'$.

In [Abd+16a], we consider qualitative verification problems for VASS-MDPs, taking as objectives control-state reachability and repeated control-state reachability. In order to define formally these problems, we need some preliminary definitions. Let $S = (Q, Q_1, Q_P, \mathtt{C}_n, \Delta, \tau)$ be a VASS-MDP and $M_S = (\Gamma, \Gamma_1, \Gamma_P, \rightarrow, prob)$ it associated MDP. For a control state $q_F \in Q$, we denote by $[\![\mathtt{F} q_F]\!]$ the set of maximal plays $\{\gamma_0\gamma_1\gamma_2 \ldots \in \Omega \mid \exists i. \gamma_i = (q_F, \mathbf{v}) \text{ for some } \mathbf{v} \in \mathbb{N}^n\}$ for which there exists a reachable configuration having q_F as a control state. Similarly, we denote by $[\![\mathtt{GF} q_F]\!]$ the set of maximal plays $\{\gamma_0\gamma_1\gamma_2 \in \Omega \ldots \mid \{i \in \mathbb{N} \mid \gamma_i = (q_F, \mathbf{v}) \text{ for some } \mathbf{v} \in \mathbb{N}^n\} \text{ is infinite}\}$ which visits infinitely often the control state q_F. Since M_S is a MDP with a countable set of configurations, the set of maximal plays $[\![\mathtt{F} q_F]\!]$ and $[\![\mathtt{GF} q_F]\!]$ are measurable (see e.g. [BK08]).

We are now ready to introduce the six problems which all takes as input: :

- A VASS-MDP $S = (Q, Q_1, Q_P, \mathtt{C}_n, \Delta, \tau)$,

- an initial configuration $\gamma_0 \in Q \times \mathbb{N}^n$, and,
- a control state $q_F \in Q$.

Each problem is declined in two versions, one regarding control-state reachability and the other one control-state repeated reachability. First we deal with the sure version of these problems where in a certain sense VASS-MDPs are interpreted as a 2-Player games and no probability is involved. These systems are then equivalent to VASS games as presented in the previous sections (with simpler objectives).

SURE-VASS-MDP-REACH

Question: Does there exist a scheduler π such that $\texttt{Plays}(M_S, \gamma_0, \pi) \subseteq [\![\texttt{F} q_F]\!]$?

SURE-VASS-MPD-REPREACH

Question: Does there exist a scheduler $\pi \in \Pi$ such that $\texttt{Plays}(M_S, \gamma_o, \pi) \subseteq [\![\texttt{GF} q_F]\!]$?

Then we will look at the almost-sure version of this problem where we seek for the existence of a scheduler guaranteeing with probability one the objective.

ALMOSTSURE-VASS-MDP-REACH

Question: Does there exist a scheduler $\pi \in \Pi$ such that $\mathbb{P}(M_S, \gamma_0, \pi, [\![\texttt{F} q_F]\!]) = 1$?

ALMOSTSURE-VASS-MPD-REPREACH

Question: Does there exist a scheduler $\pi \in \Pi$ such that $\mathbb{P}(M_S, \gamma_0, \pi, [\![\texttt{GF} q_F]\!]) = 1$?

Finally, we have the limit-sure version of these problems, where we will check whether the supremum probability on all schedulers for the concerned objective is equal to one.

LIMITSURE-VASS-MDP-REACH

Question: Does $\mathbb{P}^{sup}(M_S, \gamma_0, [\![\texttt{F} q_F]\!]) = 1$?

LIMITSURE-VASS-MPD-REPREACH

Question: Does $\mathbb{P}^{sup}(M_S, \gamma_0, [\![\texttt{GF} q_F]\!]) = 1$?

Note that sure (repeated) reachability implies, almost-sure (repeated) reachability which implies limit-sure (repeated) reachability but as we shall see the reverse implications are not true in general.

Example 6.4. *We consider the VASS-MDP S depicted on Figure 6.5 for which we suppose that the weight function assigns* 1 *to all the transitions. We take as initial configuration* $\gamma_0 = (q_0, \boldsymbol{o})$. *First we see that from* γ_0, *the system can surely reach the state* q_4, *i.e. there exists a scheduler* π *such that*

$\texttt{Plays}(M_S, \gamma_0, \pi) \subseteq [\![\texttt{F} q_4]\!]$. *In fact, this scheduler takes once the loop between* q_0 *and* q_1*s and then it moves to the configuration* $(q_2, [1, 0])$ *belonging to the probabilistic player, who will be force to move to* q_4 *after taking once the loop with* q_3 *as the value of the first counter will then be* 0*. From* γ_0 *it is not possibly to reach* q_7 *surely because of the loop between the states* q_5 *and* q_6 *but this state is almost-surely reachable, as the probability of the infinite path between* q_5 *and* q_6 *is* 0*. Finally, the state* q_{10} *is neither surely, nor almost-surely reachable, but it can be reached limit-surely, in fact the more times the scheduler loops between* q_0 *and* q_1 *at the beginning of the play, the higher is the probability to reach* q_{10} *afterwards since higher counter values for the first counter, allows him to loop more between* q_8 *and* q_9*.*

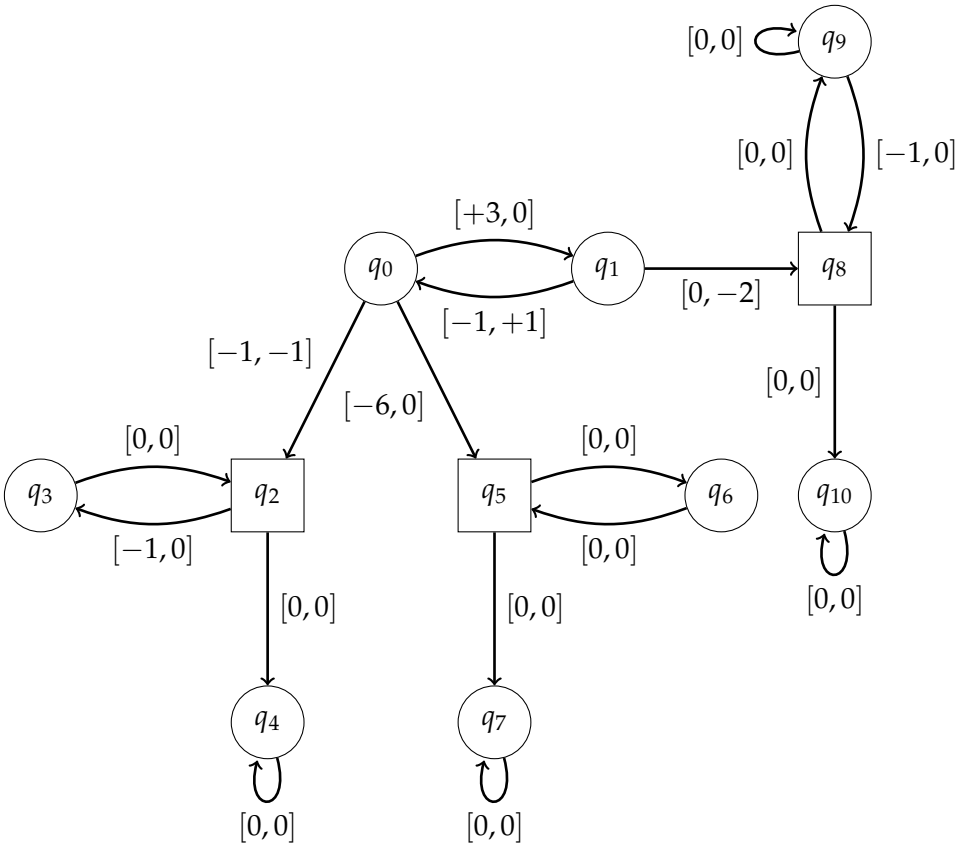

Figure 6.5: A VASS-MDP

Similarly to the undecidability result for Parity-VASS-Game, all the problems mentioned above for VASS-MDPs are undecidable. The reduction from the halting problem of deterministic Minksy machines is basically the same as the one presented on Figure 6.3 but here we consider that Player 2 is a probabilistic player and all the edges have weight 1. We shall see now that here as well restricting the ability of one of the players to modify the counters values allows to regain, in some cases, decidability.

6.3.3 *Verification of P-VASS-MDP*

First, we study the case where the Player 1 cannot modify the counters. A VASS-MDP $S = (Q, Q_1, Q_P, \mathtt{C}_n, \Delta, \tau)$ is a *P-VASS-MDP* if and only if for all $(q, true, \mathbf{b}, q') \in \Delta$ with $q \in Q_P$, we have $\mathbf{b} = \mathbf{0}$.

Unlike for single-sided Parity VASS games where we can get rid of deadlocks, we show in [Abd+16a] that the presence or absence of deadlocks in P-VASS-MDP can change the decidability status of the qualitative verifications problems. First, note that the reduction presented in Figure 6.3 does not carry over to P-VASS-MDPs, because in that construction both players have the ability to change the counter values. However, it is possible to perform a similar reduction leading to the undecidability of verification problems for P-VASS-MDP, the main difference being that we crucially exploit the fact that the P-VASS-MDP can contain deadlocks. We now explain the idea behind our encoding of Minsky machines into P-VASS-MDPs. Intuitively, Player 1 chooses a transition of the Minsky machine to simulate, anticipating the modification of the counters values, and Player P is then in charge of performing the change. If Player 1 chooses a transition with a decrement and the accessed counter value is actually 0, then Player P will be in a deadlock state and consequently the desired control state will not be reached. Furthermore, if Player 1 decides to perform a zero-test when the counter value is strictly positive, then Player P is able to punish this choice by entering a deadlock state. Player P can test if the value of the counter is strictly greater than 0 by decrementing it. The encoding of the Minsky machine is presented in Figure 6.6 where we have represented the encoding of the instructions (1) $L : \mathtt{x}_1 := \mathtt{x}_1 + 1$; `goto` L' and (2) J : `if` $\mathtt{x}_2 = 0$ `then goto` J' `else` $\mathtt{x}_2 := \mathtt{x}_2 - 1$; `goto` J'' and where each edge has a weight equal to 1. Note that no outgoing edge of Player 1's states changes the counter values. Furthermore, we see that Player P reaches the control state ☹ if and only if Player 1 chooses to take a transition with a zero-test when the value of the tested counter is not equal to 0. For the encoding of the instruction J : `if` $\mathtt{x}_2 = 0$ `then goto` J' `else` $\mathtt{x}_2 := \mathtt{x}_2 - 1$; `goto` J'', when Player P is in the control state between J and J'', it can be in a deadlock if the value of the second counter is not positive. In the sequel we will see that in P-VASS-MDP without deadlocks the sure reachability problem becomes decidable.

Theorem 6.5. *[Abd+16a] The six following problems:*

- Sure-VASS-MDP-Reach *and* Sure-VASS-MPD-RepReach,
- AlmostSure-VASS-MDP-Reach *and* AlmostSure-VASS-MPD-RepReach
- LimitSure-VASS-MDP-Reach *and* LimitSure-VASS-MPD-RepReach

are undecidable for P-VASS-MDP (of dimension 2).

However when considering deadlock-free P-VASS-MDP, we show that the sure (repeated) reachability problem is decidable. Let $S = (Q, Q_1, Q_P, \mathtt{C}_n, \Delta, \tau)$ be a deadlock-free P-VASS-MDP and $q_F \in Q$ a control state. Note that because the P-VASS-MDP S is deadlock free, the probabilistic Player P cannot take the play to a deadlock to avoid the control state q_F, but he has to deal only with infinite plays. Since S is a P-VASS-MDP, the VASS $(Q, \mathtt{C}_n, \Delta)$ is (Q_P, Q_1)-single-sided. We can use Theorem 6.3 to deduce the decidability of sure (repeated) reachability in deadlock-free P-VASS-MDP.

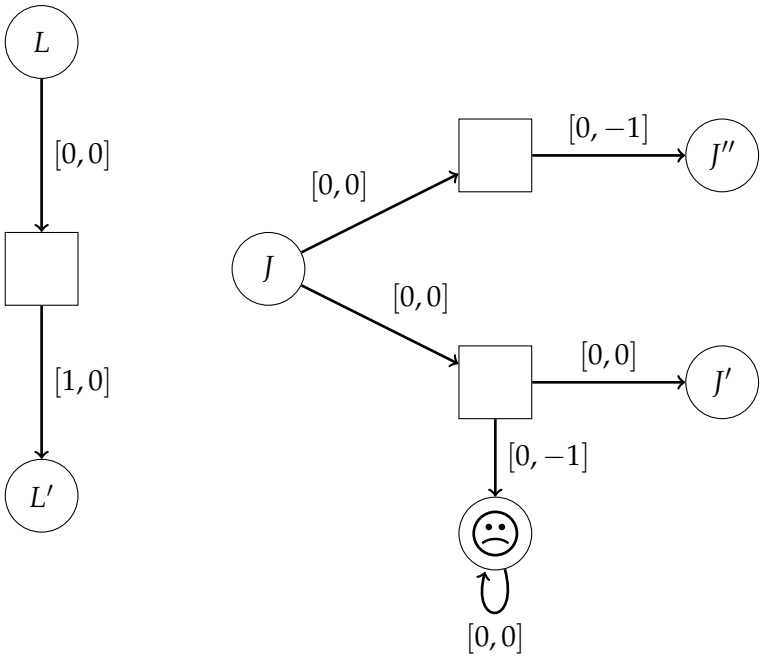

Figure 6.6: Encoding the instructions of a Minsky machine in P-VASS-MDP

Theorem 6.6. *[Abd+16a]* SURE-VASS-MDP-REACH *and* SURE-VASS-MPD-REPREACH *are decidable for deadlock-free P-VASS-MDP.*

For the corresponding almost-sure and limit-sure problems we show undecidability even when the considered P-VASS-MDP are deadlock free. We rely again on a reduction from the halting problem for deterministic Minsky machines. The main difference with the construction used for the proof of Theorem 6.5 lies in the addition of a self-loop in the encoding of the transitions for decrementing a counter, in order to avoid deadlocks. In fact, we add a self-loop with no effect on the counters on the probabilistic state between J and J'' on Figure 6.6. If Player 1, from a configuration $(J, \mathbf{v})$, chooses the transition which decrements the second counter, then the probabilistic state with the self-loop is entered, and there are two possible cases: if $\mathbf{v}[2] > 0$ then the probability of staying forever in this loop is 0 and the probability of eventually going to state J'' is 1; on the other hand, if $\mathbf{v}[2] = 0$ then the probability of staying forever in the self-loop is 1, since the other transition that leaves the state of Player P and which performs the decrement on the second counter effectively is not available. Note that such a construction does not hold in the case of sure reachability, because the path that stays forever in the loop is a valid path.

Theorem 6.7. *[Abd+16a] The four following problems:*

- ALMOSTSURE-VASS-MDP-REACH *and* ALMOSTSURE-VASS-MPD-REPREACH
- LIMITSURE-VASS-MDP-REACH *and* LIMITSURE-VASS-MPD-REPREACH

are undecidable for deadlock-free P-VASS-MDP (of dimension 2).

6.3.4 *Verification of 1-VASS-MDP*

We shall now see that when we restrict the power of the probabilistic player then different results can be obtained. A VASS-MDP $S = (Q, Q_1, Q_P, \mathtt{C}_n, \Delta, \tau)$ is a *1-VASS-MDP* if and only if for all $(q, true, \mathbf{b}, q') \in \Delta$ with $q \in Q_1$, we have $\mathbf{b} = \mathbf{0}$.

First, oppositely to the case of P-VASS-MDP, in [Abd+16a] we show that for 1-VASS-MDP we can get rid of deadlocks without changing the status of the qualitative verification problems, consequently we can assume that the considered 1-VASS-MDPsare deadlock free.

For (repeated) sure reachability, similarly to Theorem 6.6 for P-VASS-MDP, we can consider a 1-VASS-MDP as a single-sided Parity VASS game and obtain decidability using Theorem 6.3.

Theorem 6.8. *[Abd+16a]* Sure-VASS-MDP-Reach *and* Sure-VASS-MPD-RepReach *are decidable for 1-VASS-MDP.*

For what concerns the almost sure qualitative problems, we have proven that given a control state q_F we can compute the set of configurations from which it is possible to almost-surely (repeatedly) reach q_F. For this, we rely on the fact that 1-VASS-MDP can be interpreted as single-sided VASS and that such set of configurations can be express in L_μ^{sv}. Theorem 6.4 allows us then to conclude. Note that it is well-known for finite state systems, that such a set of configurations can be represented by a μ-calculus formula, see e.g. [Cha+09], but this characterisation does not automatically extend to infinite state systems, and in our case it works well because the underlying system is a VASS.

We consider a 1-VASS-MDP $S = (Q, Q_1, Q_P, \mathtt{C}_n, \Delta, \tau)$, its associated MDP $M_S = (\Gamma, \Gamma_1, \Gamma_P, \rightarrow, prob)$ and a control state $q_F \in Q$. We denote by $\mathtt{AS-Reach}$ the set $\{\gamma \in \Gamma \mid \exists \pi \in \Pi.\mathbb{P}(M_S, \gamma, \pi, [\![\mathtt{F} q_F]\!]) = 1\}$ of configurations from which q_F can almost-surely be reached and by $\mathtt{AS-RepReach}$ the set $\{\gamma \in \Gamma \mid \exists \pi \in \Pi.\mathbb{P}(M_S, \gamma, \pi, [\![\mathtt{GF} q_F]\!]) = 1\}$ the set of configurations from which we can almost-surely repeatedly reach q_F. In [Abd+16a], we have shown the following equalities:

1. $\mathtt{AS-Reach} = [\![\nu X.\mu Y.(q_F \vee \mathtt{InvPre}(X, Y))]\!]$
2. $\mathtt{AS-RepReach} = [\![\nu X.\mathtt{InvPre}(X, \mu Y.(q_F \vee \mathtt{InvPre}(\mathtt{X}, \mathtt{Y})))]\!]$

where $\mathtt{InvPre}(X, Y) = (Q_1 \wedge \Diamond(X \wedge Y)) \vee (\Diamond Y \wedge Q_P \wedge \Box X)$. Intuitively this latter formula, which belongs to L_μ^{sv} for the (Q_1, Q_P)-single-sided VASS $(Q, \mathtt{C}_n, \Delta)$, represents the set of configurations from which Player 1 can make a move towards the set represented by the intersection of the sets characterised by X and Y and Player P can make a move to the set represented by Y and cannot avoid making a transition to the set represented by X. Note that to show that these two equalities are correct, we rely on the fact that there exists $N \in \mathbb{N}$ and a scheduler such that if a configuration belongs to $[\![\nu X.\mu Y.(q_F \vee \mathtt{InvPre}(X, Y))]\!]$, then the state q_F can be reached in less than N steps and the probabilistic player cannot bring the play outside of $[\![\nu X.\mu Y.(q_F \vee \mathtt{InvPre}(X, Y))]\!]$. This result relies on the fact that this set is upward-closed and can be computed by an iterative sequence of upward-closed sets, each one included in the successor. As a matter of fact, our proof technique does not apply

for general infinite-state MDPs. Since the formulas to characterise the sets $\mathtt{AS}-\mathtt{Reach}$ and $\mathtt{AS}-\mathtt{RepReach}$ belong to the guarded fragment of the μ-calculus L_μ^{sv} interpreted over the (Q_1, Q_P)-single-sided VASS $(Q, \mathtt{C}_n, \Delta)$, Theorem 6.4 allows us to obtain the following result.

Theorem 6.9. *[Abd+16a]* AlmostSure-VASS-MDP-Reach *and* AlmostSure-VASS-MPD-RepReach *are decidable for 1-VASS-MDPs.*

Finally, we obtain as well a decidability result for what concerns the limit-sure reachability problems. For this we rely on a technique similar to the one used to solve single-sided Parity VASS game relying on a construction inspired by the one proposed by Karp and Miller [KM69] to build a finite tree representing an abstraction of the possible executions of a VASS.

Theorem 6.10. *[Abd+16a]* LimitSure-VASS-MDP-Reach *is decidable for 1-VASS-MDPs.*

6.3.5 *Summary of the results*

	P-VASS-MDP	df P-VASS-MSP	1-VASS-MDP
Sure-VASS-MDP-Reach	Undecidable [Thm 6.5]	Decidable [Thm 6.6]	Decidable [Thm 6.8]
Sure-VASS-MPD-RepReach	Undecidable [Thm 6.5]	Decidable [Thm 6.6]	Decidable [Thm 6.8]
AlmostSure-VASS-MDP-Reach	Undecidable [Thm 6.5]	Undecidable [Thm 6.7]	Decidable [Thm 6.9]
AlmostSure-VASS-MPD-RepReach	Undecidable [Thm 6.5]	Undecidable [Thm 6.7]	Decidable [Thm 6.9]
LimitSure-VASS-MDP-Reach	Undecidable [Thm 6.5]	Undecidable [Thm 6.7]	Decidable [Thm 6.10]
LimitSure-VASS-MPD-RepReach	Undecidable [Thm 6.5]	Undecidable [Thm 6.7]	Open

Table 6.1: Decidability of the qualitative verification of VASS-MDP

The Table 6.1 sums up the results we have obtained in [Abd+16a] for the qualitative verification of VASS-MDP. In this table, the abbreviation df stands for deadlock-free. For these problems, we did not establish any complexity bounds. However, some upper bounds can be retrieve using the fact that solving Parity VASS-game is a 2EXPTIME-complete problem[Col+17]. Since most of the decidable results rely on a translation towards such games (sometimes through the model-checking of μ-calculus). Finally, the decidability of the limit sure repeated reachability problem for 1-VASS-MDP is an open problem. A hint of its difficulty is given by the fact that there are instances where the property holds

even though a small chance of reaching a deadlock cannot be avoided from any reachable configuration. In particular, a solution would require an analysis of the long-run behavior of multi-dimensional random walks induced by probabilistic VASS. However, these may exhibit strange nonregular behaviours for dimensions greater than 3 as described in [Brá+15].

Part II

VERIFICATION OF PARAMETERISED NETWORKS

In this part, I will detail the results I obtained on the verification of parameterised networks. Such systems are defined by a protocol which is executed by all the entities of a network. The difficulty for the verification process in this setting lies in the fact that the size of the network is fixed but a priori unknown. Hence one needs to establish results which are somehow independant of the number of entities executing the protocol. Most of the studied verification problems seek for a number of participants which allows to witness a certain property. Because of the parameterised number of participants, this class of systems can be seen as well as infinite-state systems; in fact the whole system consists in the union of each system with a fixed number of participants. The main mean of communication I considered is inspired by ad-hoc networks and it can be defined as a broadcast over a graph of connectivity (with or without mobility). These works lie in the direct continuity of the work of German and Sistla [GS92] which deals with the verification of networks of entities communicating thanks to rendez-vous.

7

VERIFICATION OF AD HOC NETWORKS

In this chapter, we present the model of parameterised networks we have introduced in [DSZ10]. In this model, we assume that each entities executes the same broadcast protocol, given by a finite state automaton, and the communication is performed thanks to a broadcast to the neighbourhood. In order to model this last characteristic, we represent the network as an undirected graph and an edge between two entities expresses that they are neighbours in the network. In this first model, we furthermore impose that an entity which can receive a broadcast, cannot ignore it if it is performed, however it is not mandatory that an entity has to always be able to receive a broadcast.

Our model is inspired by real Ad Hoc Networks where the communication is performed by radio transmission and all the entities in the range of an emitter receive the emitted message. We assume in this chapter that the communication topology does not evolve during time, in other words, all the entities always have the same neighbours and during an execution, new entities cannot appear in the network and no entity can disappear. Furthermore this model was inspired by the ω-calculus [SRS09], a formal model of Ad Hoc Networks with selective broadcast and spontaneous movement. In the ω-calculus a configuration consists of a finite set of processes. Each process has a local state and an interface containing a finite set of group names. A group name represents a possible communication link with other processes in the network. From an abstract point of view, the structure underlying a configuration of the ω-calculus is a finite graph that defines the communication topology of a network. A node in the graph represents the current state of an individual process. There exists an edge between two nodes if the corresponding interfaces share a common group name. Adjacent nodes are called single-hop neighbours. Processes communicate through selective broadcast. Specifically, a broadcast message can be received only by the set of single-hop neighbours of the emitter. Even if we do not consider here mobility, we shall see in the next chapter how we can add this feature in our model and as well that it has important consequences for the verification process.

When the number of nodes is fixed a priori, formal models of Ad Hoc Networks like those provided by the ω-calculus can be verified by using finite-state model checking or constraint-based model-checking as done in [SRS09]. We study here the case in which networks have arbitrary size and possibly unknown topology and for this matter there is a need for new methods. Studying such parameterised systems is however not a new thematic and another work from which we had inspiration is the seminal paper by German and Sistla [GS92] where the authors study parameterised networks in which the nodes communicate thanks to pairwise rendez-vous. Later on in [EFM99], networks with broadcast communication have been studied, the main difference with our method being the absence of communication topology, i.e. when a broadcast is performed all the entities in the networks that can receive it performs the reception. This work can be retrieved in our formalisation by restricting our systems to communication topologies which correspond to complete graphs (in which between each pair of nodes there is an edge).

CONTRIBUTIONS. In [DSZ10], we have first introduced the model of parameterised Ad Hoc Networks and we have studied two main reachability questions on this model. The first one asks given a broadcast protocol whether there is a communication topology for which we can find an execution leading an entity to a specific control state. The second one can be formulated similarly, except that it asks whether a configuration can be reached where all the entities are in a specific control state. The difficulty in these problems lies in finding the initial communication topology. In fact, when a topology is provided, since it fixes both the number of entities and the way they communicate with each other, the verification boils down to the analysis of a finite state system. Unfortunately, we showed that the two considered problems are undecidable. However we have shown that decidability can be regained for the first above mentioned problem by restricting the allowed communication topologies. We obtained a first decidability result in [DSZ10] which we refined later on in [DSZ11]. To obtain these two results, we relies on the theory of well-structured transition systems [Abd+96; Abd+00; FS01] and to apply this theory we found in [DSZ11] a new class of graphs equipped with a well-quasi-order which extends in a significant way other well-known well-quasi-orders on graphs.

In [Abd+11; Abd+16b], we proposed an extension of our model of parameterised Ad Hoc Networks by making our protocols time sensitive. For this, we equipped each entity with a set of clocks, the same way clocks are added to finite state automata to obtain timed automata [AD94]; each clock evolves at the same time rate and each entity can compare the values of its clocks and reset them. We followed here the same path as in [AJ03] where the authors have extended the model of parameterised networks with rendez-vous communication proposed in [GS92] by adding clocks to the considered protocols. We have shown that we can in fact use similar techniques developed for timed networks with rendez-vous to obtain new results on the verification of timed Ad Hoc Networks. The verification problem we studied in this work consists again in seeking for an initial configuration from which another configuration exhibiting a specific control state can be reached. Since this model is richer than the model without time constraints, this problem is as well undecidable for timed Ad Hoc Networks and we provided here again some restrictions (on the shape of the communication toplogies and the number of allowed clocks) allowing to regain decidability.

7.1 BROADCAST PROTOCOLS AND AD HOC NETWORKS

7.1.1 *Broadcast protocols*

We consider networks of entities, that we will call equivalently nodes or processes, which all execute the same protocol given by a finite state system. We hence begin by providing the definition of such protocols. Given a finite alphabet of messages M, we use the notations $\texttt{BAct}(M)$ to represent the set of 'Broadcast Actions' corresponding to $\{!!m, ??m \mid m \in M\} \cup \{\tau\}$. Intuitively, τ is used for an internal action of an entity, i.e. an action that is done independently, $!!m$ corresponds to the emission of the message m and $??m$ corresponds to the reception of the message m.

Definition 7.1 (Broadcast Protocol)**.** *A* Broadcast Protocol *BP is a tuple* (Q, M, Δ, q_{in}) *where:*

- Q *is a finite set of control states,*
- M *is a finite alphabet of messages,*

- $\Delta \subseteq Q \times \texttt{BAct}(M) \times Q$ *is a finite set of edges labelled by broadcast actions, and,*
- $q_{in} \in Q$ *is the initial control state.*

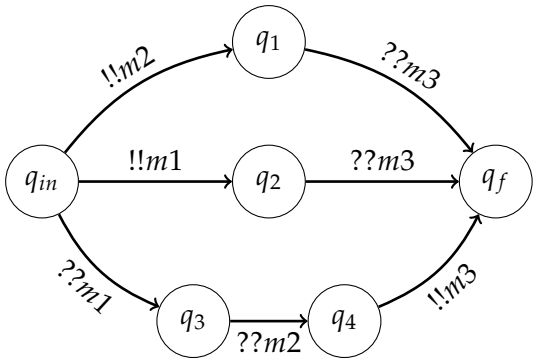

Figure 7.1: A Broadcast Protocol

Figure 7.1 provides an example of a Broadcast Protocol with six control states and three types of messages that can be broadcasted ($m1$, $m2$ and $m3$).

7.1.2 *Ad Hoc Network induced by a protocol*

We how now present the ad hoc networks configurations. In these networks, we assume that each entity can only communicate with its neighbours and each entity is in a state of the considered broadcast protocol. Hence, we use labeled graphs to represent such configurations where the nodes/vertices of the graph correspond to the entities of the network and an edge between two nodes expresses the fact that the nodes are neighbours.

Definition 7.2 (Q-graphs)**.** *Given a set of elements Q, a* Q-graph *is a labelled undirected graph $\gamma = (V, E, L)$ where:*

- *V is a finite set of nodes,*
- *$E \subseteq V \times V \setminus \{(v, v) \mid v \in V\}$ is a finite set of edges such that $(u, v) \in E$ implies $(v, u) \in E$, and,*
- *$L : V \mapsto Q$ is a labeling function.*

We use $L(\gamma)$ to denote the set $\{q \in Q \mid \exists v \in V.L(v) = q\}$ of labels present in the Q-graph γ. Given two nodes $u, v \in V$, we will write $u \sim_\gamma v$ if and only if we have $(u, v) \in E$ (the nodes u and v are adjacent in γ) and we might omit γ and write simply $u \sim v$ when the considered Q-graph is clear from the context.

We are now ready to move to the definition of an Ad Hoc Networks induced by a broadcast protocol $BP = (Q, M, \Delta, q_{in})$. It is given in term of a transition system. The set of configurations, denoted by Γ, is the set of Q-graphs and the set of initial configurations $\Gamma_{in} \subseteq \Gamma$ is the set of $\{q_{in}\} - graphs$. On Figure 7.2, a configuration with six nodes for the broadcast protocol of Figure 7.1 is depicted.

Given a Q-graph $\gamma = (V, E, L)$, a node $v \in V$ and a message $m \in \Sigma$, we define the set $R^\gamma_m(v) = \{v' \in V \mid v \sim v' \text{ and } \exists q' \in Q.(L(v'), ??m, q') \in \Delta\}$ characterising the neighbors of $v \in \gamma$ which can receive the message m. The transition system $AHN(BP)$ induced by

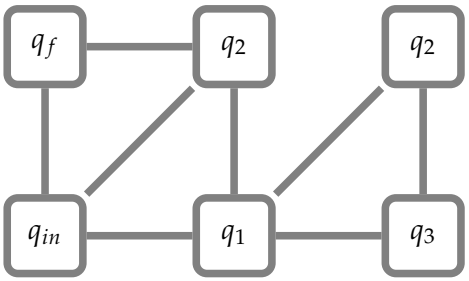

Figure 7.2: A possible configuration for the protocol of Figure 7.1

BP is then given by the tuple $(\Gamma, \rightarrow)$ where $\rightarrow \subseteq \Gamma \times \Delta \times \Gamma$ is the transition relation which we shall now define. We have $\gamma \xrightarrow{\delta} \gamma'$ if, and only if, $\gamma = (V, E, L)$ and $\gamma' = (V, E, L')$ and $\delta = (q, a, q')$ and one the following conditions holds:

- **Internal action:** $a = \tau$ and there exists $v \in V$ such that $L(v) = q$ and $L'(v) = q'$ and $L(v) = L'(v')$ for all $v' \in V \setminus \{v\}$;
- **Broadcast communication:** $a = !!m$ and there exists $v \in V$ such that $L(v) = q$ and $L'(v) = q'$ and $(L(v'), ??m, L'(v')) \in \Delta$ for all $v' \in R_m^\gamma(v)$ and $L'(v') = L(v')$ for all $v' \in V \setminus R_m^\gamma(v)$.

We observe that the graph structure does not change when taking a transition, but only the labels of the nodes evolve. Furthermore, an internal action has effect on a single node and when a broadcast $!!m$ is performed by a node v, it is delivered only to the subset of neighbors which can receive it (and have to receive it). We denote by $\gamma \rightarrow \gamma'$ if there exsits $\delta \in \Delta$ such that $\gamma \xrightarrow{\delta} \gamma'$ and $\rightarrow^*$ represents the reflexive and transitive closure of $\rightarrow$. Given an initial configuration $\gamma_{in} \in \Gamma_{in}$, a finite run (or finite execution) starting from γ_{in} in $AHN(BP)$ is a finite path in $AHN(BP)$ denoted as:

$$\rho := \gamma_0 \xrightarrow{\delta_0} \gamma_1 \xrightarrow{\delta_1} \dots \xrightarrow{\delta_{n-1}} \gamma_n$$

with $\gamma_0 = \gamma_{in}$.

Example 7.1. *Figure 7.3 provides an example of execution for the Broadcast Protocol of Figure 7.1. In each configuration, we have indicated the node which was responsible of the broadcast. For instance, at the first stage, the node located at the bottom left takes the translation $(q_{in}, !!m1, q_2)$ and as a consequence, all its neighbours in state q_{in} (from which $m1$ can be received) move to q_3.*

7.1.3 *Reachability problems*

We introduce the two verification problems we have studied for Ad Hoc Networks induced by Broadcast Protocols. These problems are both formulated in the parameterised case in which the size and the topology of the networks are not known. The first problem is the parameterised control state reachability which asks whether a configuration exhibiting a specific control state can be reached from an initial configuration. It can be formulated as follows:

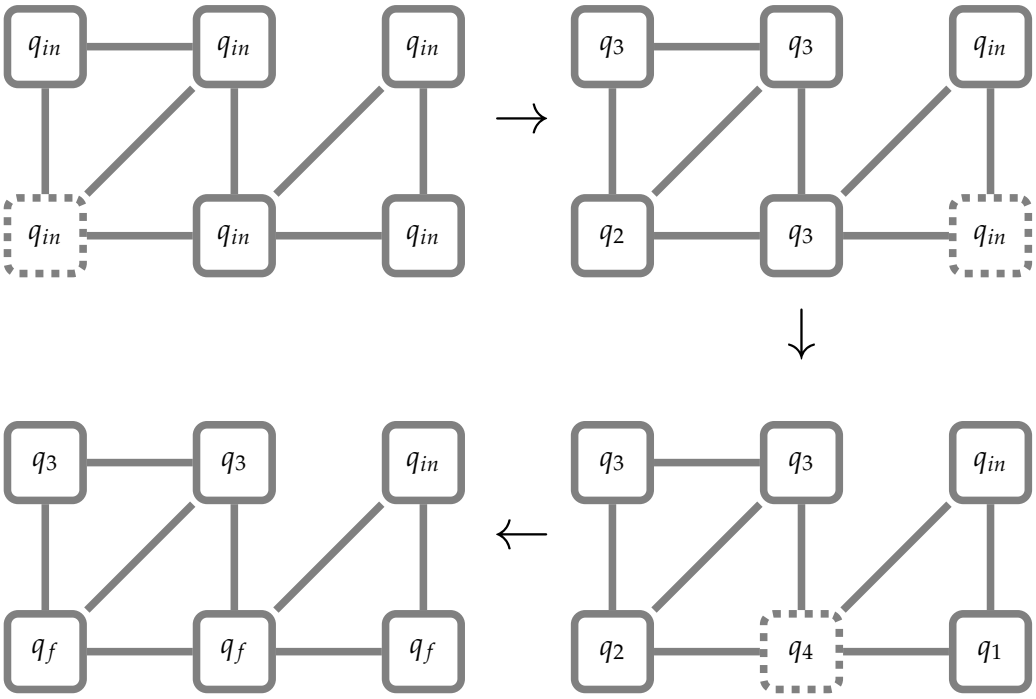

Figure 7.3: A finite run in the Ad Hoc Network associated to the protocol of Figure 7.1

AHN-ControlReach	
Input:	A Broadcast Protocol $BP = (Q, M, \Delta, q_{in})$ and a control state $q_f \in Q$;
Question:	Does there exists an initial configuration γ_{in} and a configuration γ such that $q_f \in L(\gamma)$ and $\gamma_{in} \rightarrow^* \gamma$ in $AHN(BP)$?

We point out that we do not impose here any constraints on the seeked initial configurations and this is where lies the difficulty of this problem. Indeed, since in Ad Hoc Networks the communication topology does not change (but only the labels of the different entities evolve), if one fixes an initial configuration γ_{in}, then the number of reachable configurations from γ_{in} is finite, but the set of initial configurations is infinite. To justify the interest of this verification problem, one can think of the control state q_f as a bad state of the protocol that should never be seen in any reachable configurations and if the answer to AHN-ControlReach is positive then it means that this specification is not respected.

The second problem we have studied is very similar, the main difference being that its asks to reach a configuration where all the nodes are in the control state q_f of the protocol. We call this problem, target state reachability and define it as follows:

AHN-TARGET

Input: A Broadcast Protocol $BP = (Q, M, \Delta, q_{in})$ and a control state $q_f \in Q$;

Question: Does there exists an initial configuration γ_{in} and a configuration γ such that $L(\gamma) = \{q_f\}$ and $\gamma_{in} \rightarrow^* \gamma$ in $AHN(BP)$?

Imagine that the control state q_f represents a deadlock state in the Broadcast Protocol, then this problem basically asks whether there is an initial configuration from which an execution will lead to a configuration where all the nodes are in this deadlock state. Note that if we consider the same Broadcast Protocol and the same control state in the two previously mentioned protocols, then a positive answer to AHN-TARGET implies a positive anwer to AHN-CONTROLREACH.

Example 7.2. *If we consider the Broadcast Protocol depicted on Figure 7.1 with the control state q_f then the answer to the problems* AHN-TARGET *is positive, indeed a witness execution can be built by taking the same execution as the one represented on Figure 7.3 but restricting the configurations to the three nodes at the bottom.*

7.2 UNDECIDABILITY IN THE GENERAL CASE

In [DSZ10], we have first proven that AHN-CONTROLREACH and AHN-TARGET are undecidable. To obtain this result, we have performed a reduction from the halting problem for deterministic Minsky machines (see Section 3.2). The main difficulty in Ad Hoc Networks to simulate a Minsky machine lies in the fact that we can have any initial configuration and this lack of structure renders tough a simulation. To overcome this difficulty, we begin to show that we can in a certain sense extract a structure from any initial configuration, by showing that if a certain state is reachable from a given initial configuration, then in the reached configuration, some nodes are arranged in a specific way (in order for instance to form a line) and the other nodes do not participate anymore to the communcation (they are sent in a deadlock state).

To illustrate how we can extract a structure from a communication topology, we have built in [DSZ10] a specific Broadcast Protocol depicted on Figure 7.4 where the states q_0 and p_0 are reachable from the initial control state q_{in} thanks to an internal action. This protocol respects the following property: if at some point a configuration γ is reached with a node labelled by p_3, then we are sure that in γ, each node v labelled by p_3 has an unique neighbor v' label by q_3 and all the other nodes neighbors of v or v' are in the deadlock state err.

Using an extension of the RAO protocol, we can define a Brodacast Protocol which simulates the execution of a deterministic Minsky machine. This allows us to reduce the halting problem to AHN-CONTROLREACH. We now explain the intuition behind this reduction. In a first phase, we adapt the RAO protocol to ensure that a given control node is connected to two distinct lists of nodes used to simulate the contents of the two counters. Each node in the list associated to a counter x_i is either in state Z_i or NZ_i, the current value of x_i being given by the number of nodes in state NZ_i in the list. The length of each list is guessed non-deterministically during the execution of the first phase and it should be bigger than the maximum value stored in a counter for the simulation to succeed. Initially all nodes simulating counter values are in state Z_i to state that the initial value of the counters

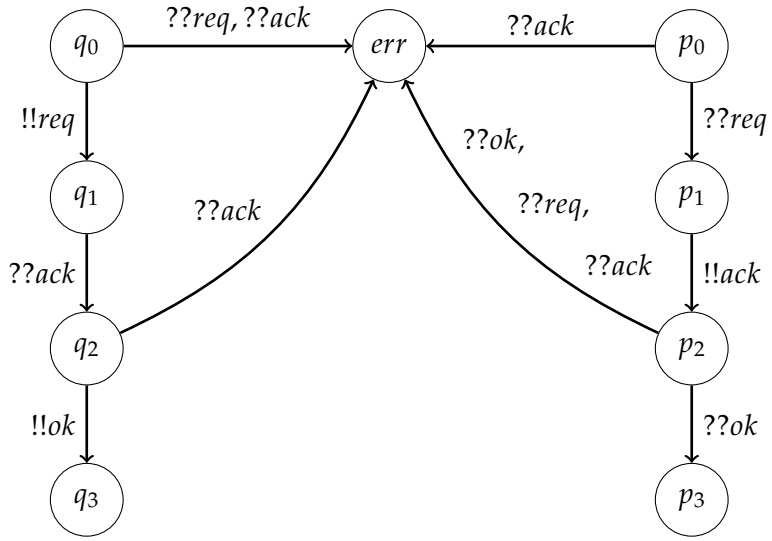

Figure 7.4: The RAO (Req/Ack/Ok) Broadcast Protocol

is zero. Since the RAO protocol can only be used to connect pair of nodes, in our extension we use three control states Z_i, Z_i' and Z_i'' to encode Z_i; this allows us to guarantee that each node in the list has only a single successor and a single predecessor (for instance, each node Z_i has a predecessor Z_i'' and a successor Z_i' in the list) and all the other nodes, if present, are sent to an 'error' state as in the RAO protocol. In the second phase, the control node starts the simulation of the instructions. It operates by querying and changing the states of the nodes in the two lists according to the type of instructions to be executed. The queries are propagating back and forth in the lists to change the states accordingly. For instance, if the instruction consists in incrementing the counter x_1, the control node sends a message that should reach the end of the list (the last node in state NZ_i), when its successor in state Z_i receives this messages it stop the propagation, changes its state to NZ_i and sends backward in the list a message to acknowledge that the increment has been correctly performed.

We show as well how to reduce AHN-TARGET to AHN-CONTROLREACH. This allows us to present our first negative result.

Theorem 7.1. *[DSZ10]* AHN-CONTROLREACH *and* AHN-TARGET *are undecidable.*

7.3 RESTRICTING THE COMMUNICATION TOPOLOGY TO REGAIN DECIDABILITY

In [DSZ10; DSZ11] we have shown that the decidability of the two problems AHN-CONTROLREACH and AHN-TARGET can be regained by restricting the set of configurations. To obtain there results, we rely on the theory of Well-Structured Transition Systems [Abd+00; Abd+96; FS01].

7.3.1 *Well-Structured Transition Systems everywhere*

We recall here some aspects of the theory of Well-Structured Transition Systems introduced in [Abd+00; Abd+96; FS01]. This theory proposes a methodology to solve some verification

problems on systems for which the set of configurations can be associated to a well-quasi-order (wqo).

Well-Structured Transition Systems correspond to transition systems equipped with a well-quasi order on the configurations which respects some properties. In the sequel, we call a transition system a tuple $(\Gamma, \rightarrow)$ where Γ is the set of configurations and $\rightarrow \Gamma \times \Gamma$ corresponds to the transition relation. Given a quasi-order $(\Gamma, \leq)$ on the configurations, the relation $\rightarrow$ is said to be *monotonic* with respect to $\leq$ if and only if it respects the following condition:

- for all $\gamma_1, \gamma_2, \gamma_1' \in \Gamma$, if $\gamma_1 \rightarrow \gamma_2$ and $\gamma_1 \leq \gamma_1'$, then there exists $\gamma_2' \in \Gamma$ such that $\gamma_1' \rightarrow \gamma_2'$.

This notion allows us to recall the definition of Well-Structured Transition Systems.

Definition 7.3. *[FS01] A Well-Structured Transition System (WSTS) is a tuple $(\Gamma, \rightarrow, \leq)$ verifying the following properties:*

- *$(\Gamma, \rightarrow)$ is a transition system,*
- *$(\Gamma, \leq)$ is a wqo,*
- *$\rightarrow$ is monotonic with respect to $\leq$.*

As shown in [FS01], many infinite state systems can be seen as WSTS as for instance the transition systems induced by Petri Nets (or equivalently VASS), by Lossy Channel Systems (with the subword ordering) or by some family of counter systems with incrementing errors.

The introduction of WSTS is motivated by the fact that in many cases some safety verification problems become decidable in these systems. We will provide here the conditions we use in the sequel to obtain decidability results for some specific WSTS. We assume that $(\Gamma, \rightarrow, \leq)$ is a WSTS. Given a subset of elements $\Gamma' \subseteq \Gamma$, we define its predecessor set, denoted by $Pred(\Gamma')$, as the set of configurations $\{\gamma \in \Gamma \mid \exists \gamma' \in \Gamma'. \gamma \rightarrow \gamma'\}$. Now thanks to the monotonicity of $\rightarrow$, one can show that if $U \subseteq \Gamma$ is an upward-closed set, then $Pred(U)$ is upward-closed. Furthermore thanks to Lemma 2.1, we know that U and $Pred(U)$ have a finite basis. One important property consists in being able to compute the finite basis of $Pred(U)$ from the finite basis of U. Following [FS01], we say that $(\Gamma, \rightarrow, \leq)$ has an *effective pred-basis* if given a finite set of elements $B \subseteq \gamma$, one can effectively compute a finite set of elements B' such that $\uparrow B' = Pred(\uparrow B)$. Finally, we say that the wqo $(\Gamma, \leq)$ is decidable if one can decide given two elements $\gamma, \gamma' \in \Gamma$ whether $\gamma \leq \gamma'$ holds or not.

These two last conditions allow to compute a finite basis for the set $Pred^*(\uparrow B) = \{\gamma \in \Gamma \mid \exists \gamma' \in \uparrow B. \gamma \rightarrow^* \gamma'\}$ (where $\rightarrow^*$ represents the reflexive and transitive closure of $\rightarrow$) and B is a finite subset of Γ. The technique to compute $Pred^*(\uparrow B)$ consists in considering an increasing (with respect to inclusion) sequence of upward-closed sets $(U_i)_{i \in \mathbb{N}}$ as follows :

- $U_0 = \uparrow B$,
- $U_{i+1} = U_i \cup Pred(U_i)$ for all $i \in \mathbb{N}$.

together with a sequence of finite set $(B_i)_{i \in \mathbb{N}}$ such that $U_i = \uparrow B_i$ for all $i \in \mathbb{N}$. Then using Lemma 2.2, we know that there exists $k \in \mathbb{N}$ such that $U_i = U_k$ for all $i \geq k$ and consequently we have $U_k = Pred^*(\uparrow B)$. Finally if $(\Gamma, \rightarrow, \leq)$ has an *effective pred-basis* and $\leq$ is decidable then it is possible to effectively compute B_k the basis of U_k.

Proposition 7.1. *[FS01] Let $(\Gamma, \to, \leq)$ be a WSTS with an effective pred-basis and such that $\leq$ is decidable. For any finite set $B \subseteq \Gamma$, one can compute a finite set $B' \subseteq \Gamma$ such that $\uparrow B' = Pred^*(\uparrow B)$.*

This proposition will be particularly useful in our context to compute the set of configurations from which an upward closed set of configurations can be reached. In fact, for AHN-ControlReach, we want to check if there is an initial configuration from which a configuration exhibiting a specific control state q_f can be reached. But we shall see that, for some specific wqo, the set of configurations exhibiting q_f can bes defined as an upward closed set of configurations for which we can provide the finite basis. Hence the problem boils down at computing the set of predecessors and checking whether it contains an initial configuration.

7.3.2 *Restrictions on the topology*

As we have seen with Theorem 7.1, AHN-ControlReach is undecidable but we will see here that we can propose some well-quasi-orders on a subset of configurations of Ad-Hoc-Networks such that the induced transition system (limited to these configurations) is a WSTS which respects the conditions of Proposition 7.1. The main difficulty boils down to finding a wqo on a subset of graphs for which the transition relation of Ad Hoc Networks happens to be monotonic. For instance, the graph minor relationship which was shown by Robertson and Seymour to be a wqo [RS04] is not well suited for our reasoning because the transition relation is not monotonic for this order.

7.3.2.1 *K-path-bounded configurations*

The first wqo we found in [DSZ10] that leads to a WSTS is the induced subgraph relation which have been shown by Ding [Din92] to be a wqo for the classes of K-bounded path graphs (where K belongs to $\mathbb{N}$). We rephrase this result in our context and what it implies for the verification Ad Hoc Networks.

We consider a Broadcast Protocol $BP = (Q, M, \Delta, q_{in})$. For a configuration $\gamma = (V, E, L)$, a *simple path* in γ is a finite sequence of vertices $v_1 v_2 \dots v_\ell$ such that $(v_i, v_{i+1}) \in E$ for all $i \in [1, \ell - 1]$ and $v_i \neq v_j$ for all $i, j \in [1, \ell]$. The length of this simple path is ℓ. Given a natural $K \geq 1$, a configuration $\gamma = (V, E, L)$ is *K-path-bounded* if and only if the length of the longest simple path in γ is at most K. We denote by $\Gamma_{K\_\texttt{Path}}$ the set of K-path-bounded configurations and if $AHN(BP) = (\Gamma, \to)$ then $K\_\texttt{Path}\_AHN(BP)$ is the trasition system restricted to K-path-bounded configuration $(\Gamma_{K\_\texttt{Path}}, \to_K)$ where $\to_K$ is the restriction of $\to$ to triple in $\Gamma_{K\_\texttt{Path}} \times \Delta \times \Gamma_{K\_\texttt{Path}}$.

Example 7.3. *On Figure 7.5, we have depicted a 5-path-bounded configuration. This configuration could represent some nodes servers, carrying the label s and nodes clients labelled with a or b. Here we could add any number of clients, each one only connected to a single server, and the obtained configuration would stay 5-path-bounded.*

We now have to define the order relation we consider on configurations which corresponds to the induced subgraph relation adapted to our context. Given two configurations $\gamma_1 = (V_1, E_1, L_1)$ and $\gamma_2 = (V_2, E_2, L_2)$, we have $\gamma_1 \preceq \gamma_2$ if and only if there exists an injective function $h : V_1 \mapsto V_2$ verifying the following conditions:

1. $(v, v') \in E_1$ **if and only if** $(h(v), h(v')) \in E_2$ for all $v, v' \in V_1$, and,

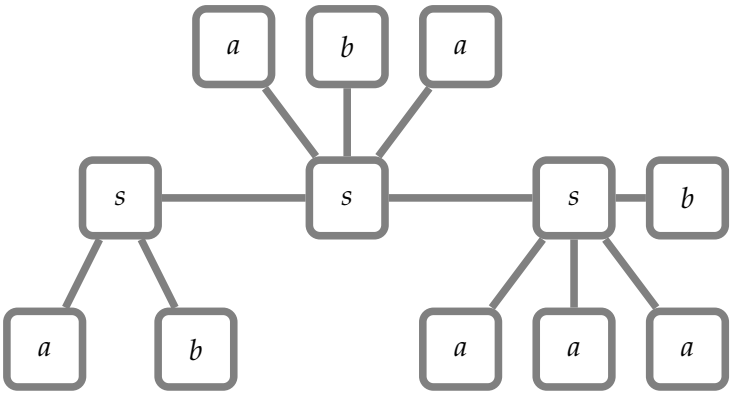

Figure 7.5: A 5-bounded configuration

2. $L_1(v) = L_2(h(v))$ for all $v \in V_1$.

Remark. *The induced subgraph relation is stronger than the usual subgraph relation which in our context would only require that each edge $(v, v') \in E_1$ has a corresponding edge $(h(v), h(v')) \in E_2$. It means that if we considered the subgraph relation in E_2 we could add new edges between $(h(v), h(v'))$ that do not have a corresponding edge $(v, v') \in E_1$. We need to consider the induced subgraph relation to guarantee the monotonicity of the transition relation.*

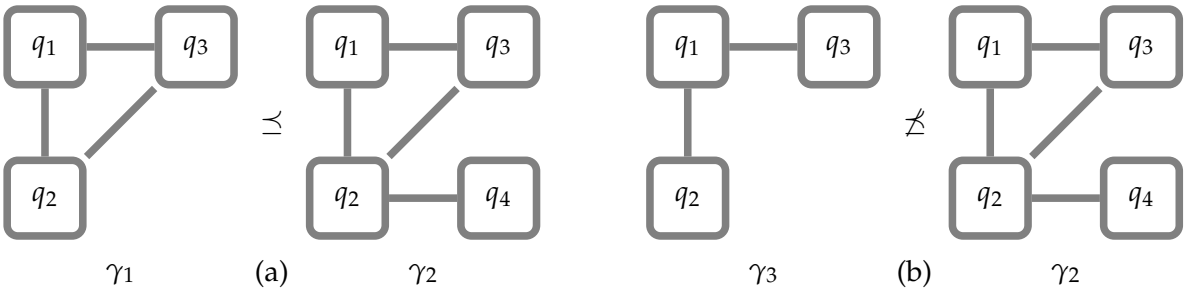

Figure 7.6: Example for the induced subgraph relation

Example 7.4. *On Figure 7.6-(a), we have depicted three configurations γ_1, γ_2 and γ_3 such that $\gamma_1 \preceq \gamma_2$ and $\gamma_3 \not\preceq \gamma_2$. In graph terminology, γ_3 is a subgraph of γ_2 but not an induced subgraph. We can furthermore justify on this example why the subgraph relation does not suffice to obtain the monotonicity of the transition relation in Ad Hoc Networks. If in γ_3, the nodes labelled with q_2 performs a broadcast, only the node in q_1 can receive it and might change its state consequently, however in γ_2, both the state in q_1 and q_3 can receive this broadcast and change their state, hence one might end in a configuration γ_2' which is not anymore a subgraph of the configuration γ_3' reached from γ_3.*

In [Din92], it is proved that for any $k \geq 1$, the subclass of k-path-bounded graphs equipped with the induced subgraph relation is a wqo. In [DSZ10], we used this result and show that, for all $K \geq 1$, $(\Gamma_{K\_\mathtt{Path}}, \rightarrow_K, \preceq)$ is a WSTS with an effective pred-basis and such that $\preceq$ is decidable. To obtain the desired decidability result, it remains to show how to encode AHN-ControlReach into a computation of $Pred^*(\uparrow B)$ for a certain finite set

of elements B. For this matter, given a control state $q_f \in Q$, we define the configuration $\gamma_f = (\{v\}, \emptyset, L_f)$ where $L_f(v) = q_f$. We can then easily show that for all $K \geq 1$ there exists an initial configuration $\gamma_{in} \in \Gamma_{K\_\mathtt{Path}}$ and a configuration γ with $q_f \in L(\gamma)$ such that $\gamma_{in} \rightarrow_K^* \gamma$ iff $Pred^*(\uparrow \{\gamma_f\}) \cap \Gamma_{in} \neq \emptyset$ in the WSTS $(\Gamma_{K\_\mathtt{Path}}, \rightarrow_K, \preceq)$. Consequently, we deduce the following result thanks to Proposition 7.1.

Theorem 7.2. *[DSZ10] For any $K \geq 1$, the problem* AHN-ControlReach *restricted to K-path-bounded configurations is decidable.*

Unfortunately, we proved as well that this result does not extend to AHN-Target where we ask that in the reached configuration all the configurations are in a given control state. Indeed we cannot use the previous technique basically because this problem does not reduce to the reachability of a configuration belonging to an upward closed set. Indeed, if we take an upward closed set with respect to the order $\preceq$, we cannot guarantee that all these configurations in this set are such that all the nodes are labelled with a single control state. We showed that this latter problem is undecidable when K-path-bounded configurations for all $K \geq 3$. The proof is very similar to the one of Theorem 7.1. It is based as well on a reduction from the halting problem for deterministic Minsky machines. The main difference being that, in this case, we cannot enforce the communication topology to have the shape of two connected lists of nodes of unbounded length (as such topologies would not be K-path-bounded for any K). Instead the communication topology we enforce, thanks to a protocol similar to the RAO protocol of Figure 7.4, are stars topology. In our context, a star of radius 1 is a graph with a central node and all other nodes (which are not in an error state and are still active in the protocol) are only connected to this central node and we call them satellite nodes. Figure 7.7 provides an example of a star of radius 1 with 5 satellite nodes.

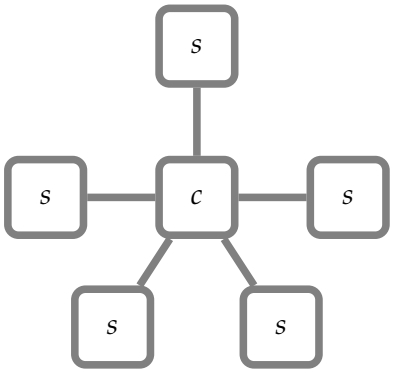

Figure 7.7: A radius 1 star with 5 satellite nodes

We design hence a protocol, which in a first phase extract a star from the topology (with an unbounded number of nodes) and in a second phase simulates the Minsky machine, the central node bein responsible for the simulation of the instructions and the satellite nodes simulating the values of the two counters. To simulate a counter, these satellite nodes are either in a state Z_i or NZ_i or *sink* and during the simulation the number of states in NZ_i indicates the values of the counter i. The increment and decrement of the counter are done in a classical way, but the most interesting part is how to test whether a counter is equal to

0 and this is where we use the fact that the considered problem is AHN-Target. Indeed, we cannot test the absence of nodes, but when performing a zero test, if some satellite nodes are in state NZ_i (meaning that the counter is not equal to 0), they are sent into the state *sink*. Consequently if at the end of the computation, there are nodes in state *sink*, this mean that at some point the controller has assumed the value of the counters was 0 but it was not the case. Hence if at the end, there is no node in state *sink*, it means that the simulation has followed correctly the rules of the Minsky machine. Consequently, only the satellite nodes not in *sink* are able to move to the final control state q_f and the hypothesis of the problem AHN-Target allows to only consider correct simulations of the machine. Thanks to this reduction and by noting that star of radius are 3-path-bounded, we can state the following negative result.

Theorem 7.3. *[DSZ10] For any $K \geq 3$, the problem* AHN-Target *restricted to K-path-bounded configurations is undecidable.*

7.3.2.2 *K-clique-path-bounded configurations*

There are other restrictions that can be imposed on the set of considered configurations in order to obtain decidability of AHN-ControlReach. For instance, in [EFM99], the authors consider a model very close to Ad Hoc Networks, the main difference being that the configurations are not equipped with a communication topology and all the entities that can receive an emitted message have to receive it. In our context, it is equivalent to consider Ad Hoc Networks restricted to complete configurations. We shall now recall this result, see how it compares with the previous results and finally show how we obtain a decidability result incorporating these two previous result.

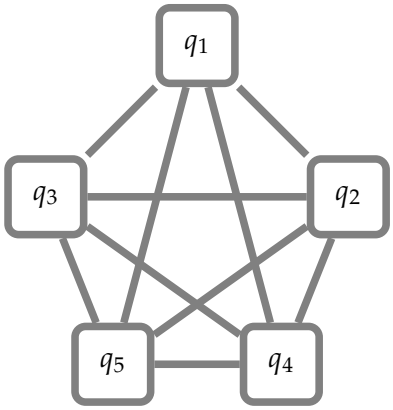

Figure 7.8: A complete configuration

We fix a Broadcast Protocol $BP = (Q, M, \Delta, q_{in})$. A configuration $\gamma = (V, E, L)$ is *complete* if, and only if, $E = V \times V \setminus \{(v, v) \mid v \in V\}$. We denote by $\Gamma_{\texttt{Comp}}$ the set of complete configurations. Figure 7.8 shows an example of a complete configuration. One can show that $(\Gamma_{\texttt{Comp}}, \preceq)$ is a wqo. In fact, for each configuration in $\Gamma_{\texttt{Comp}}$, the graph structure is always the same and can hence be forgotten, the only relevant information being, for each control state q, the number of nodes labelled by q. Consequently $(\Gamma_{\texttt{Comp}}, \preceq)$ is a wqo because it is equivalent to the quasi order $(\mathbb{N}^Q, \leq)$ which is a wqo, thanks to Dickson's lemma[Dic13].

By the same reasoning that leads to Theorem 7.2, we can retrieve the result of [EFM99] and obtain that AHN-ControlReach restricted to complete configurations is decidable.

Something that was frustrating with these two last results is that the first one does not allow to take complete graphs into account, as for any $K \geq 1$, there always exists a complete configurations γ such that $\gamma \notin \Gamma_{K\_\mathtt{Path}}$. In fact, if γ is a complete configuration with n vertices, then the length of its longest simple path is n. On the other hand, for all $K \geq 2$ there are many configurations in $\Gamma_{K\_\mathtt{Path}}$ which are not complete. The fact that configurations in $\Gamma_{K\_\mathtt{Path}}$ do not include all complete graphs or even better graphs with a complete subgraphs (of any order) is not entirely satisfying since somehow complete graphs allow to model the most optimistic view of the protocol where all the nodes can broadcast their messages to all the others and receive messages from them. However, dealing only with complete configurations would be too restrictive as it does not allow any topology. This is the reason why in [DSZ11], we introduced a new subset of configurations, which extend K-path-bounded configurations and include complete graphs, and even better every configurations which can be decomposed in a set of complete configurations connected by a bounded simple path. Not only, this result allowed us to improve the decidability result of Theorem 7.2 to larger classes of configurations, but to obtain our result, we relied on a new wqo on subset of labelled undirected graphs. We now present formally this result.

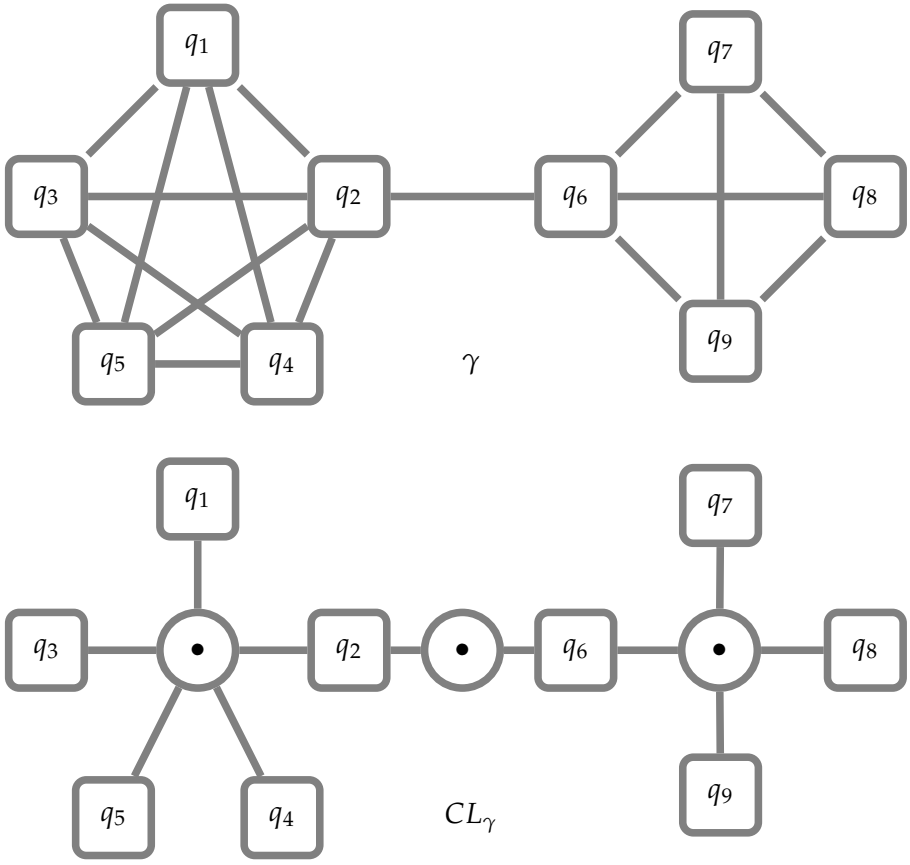

Figure 7.9: A configuration γ and its associated clique graph

For a configuration $\gamma = (V, E, L)$, the set of maximal cliques of γ is a set of subset of nodes such that if $W = \{W_1, \ldots, W_k\}$ then:

- $V = W_1 \cup \ldots \cup W_k$, and,
- for all $v, v' \in V$, we have $(v, v') \in E$ if and only if there exists $i \in [1, k]$ such that $v, v' \in W_i$, and,
- for all $i \in [1, k]$, for all $v \in W_i$, if there exists $v' \in V \setminus W_i$ such that $(v, v') \in E$, then there exists $v'' \in W_i$ such that $(v', v'') \notin E$.

Intuitively the maximal cliques of γ are the subgraphs which are complete and cannot be extended to bigger complete subgraphs and which covers E. Note that this set is unique. Based on these maximal cliques, we will propose another way to represent a configuration γ where edges between nodes belonging to the same maximal clique will be forgotten and instead another node representing the clique will be added. Let $\gamma = (V, E, L)$ be a configuration and let $\bullet \notin L(\gamma)$. We associate to γ the *clique graph* CL_γ which is the bipartite graph (V, W, E', L') such that :

- W is the set of maximal cliques of γ,
- for all $v \in V$ and $w \in W$, we have $(v, w) \in E'$ if and only if $v \in W$,
- $L'(v) = L(v)$ for all $v \in V$ and $L'(w) = \bullet$ for all $w \in W$.

Figure 7.9 provides an example of a configuration γ and its associated clique graph CL_γ. Given a natural $K \geq 1$, a configuration $\gamma = (V, E, L)$ is *K-clique-path-bounded* if and only if the length of the longest simple path in CL_γ is at most K. We denote by $\Gamma_{K\_\texttt{CliquePath}}$ the set of K-clique-path-bounded configurations. By an adaptation of the proof of Ding that shows tha the subclass of k-path-bounded graphs equipped with the induced subgraph relation is a wqo [Din92], we showed the following lemma.

Lemma 7.1. *[DSZ11]* $(\Gamma_{K\_\texttt{CliquePath}}, \preceq)$ *is a wqo for all* $k \geq 1$.

Note that this result is interesting on its own as it extends the result of Ding to larger subclasses of graphs, in fact we have that $\Gamma_{\texttt{Comp}} \subseteq \Gamma_{2\_\texttt{CliquePath}}$, which shows that complete configurations are taken into account if we consider configurations in $\Gamma_{K\_\texttt{CliquePath}}$ with $K \geq 2$, and $\Gamma_{K\_\texttt{Path}} \subseteq \Gamma_{2K\_\texttt{CliquePath}}$, consequently these subclasses of configurations allow to deal as well with K-path-bounded configurations. This lemma together with the same reasoning as for the proof of Theorem 7.2 leads us to the following decidability result.

Theorem 7.4. *[DSZ11] For any* $K \geq 1$*, the problem* AHN-ControlReach *restricted to K-clique-path-bounded configurations is decidable.*

Unfortunately, we show in [DSZ11] that for every $K \geq 2$, the problem AHN-ControlReach restricted to K-clique-path-bounded configurations is non-primitive recursive. Hence even if we have a decidable restriction for AHN-ControlReach, the complexity of the verification process is high.

7.4 TIMED EXTENSION OF AD HOC NETWORKS

In [AD94], Alur and Dill extend finite state automata by equipping the model with clocks, which correspond to variables with positive real values, that can be tested and reset by

the automaton and which are incremented at the same rate in a non deterministic fashion. This model has proven to be very useful to model time sensitive systems and its has been adapted and extended in many different ways. In particular. in [AJ03], the authors propose a model of networks with a parameterised number of entities communicating thanks to rendez-vous and where each entities is equipped with some clocks à la timed automaton. We proposed in [Abd+11; Abd+16b] a similar extension for Ad Hoc Networks that we present in this section.

7.4.1 *Timed Ad Hoc Networks*

As for Ad Hoc Networks, a Timed Ad Hoc Network (TAHN) consists of a graph where the nodes represent processes executing a common protocol defined by a timed broadcast protocol. The values of the clocks manipulated by the protocol inside each process are incremented all at the same rate. In addition, processes may perform discrete transitions which are either local transitions or communication events. When firing a local transition, a single process changes its local state without interacting with the other processes. For what concerns communication, as in Ad Hoc Networks, it is performed by means of broadcast to the neighbours, the communication topology being represented as a graph. Finally, transitions of the protocol are guarded by conditions on values of clocks and may also reset clocks. We now provide the formal definition of the model.

We assume that each process operates on a set of clocks X. In this context, a *guard* is a boolean combination of predicates of the form $x \sim k$ with $x \in X$, $\sim \in \{<\ .\ \leq, =, \geq, >, \}$ and $k \in \mathbb{N}$. We denote by $\mathtt{G}(X)$ the set of guards over X. These guards are used to impose conditions on the clocks of each process. A *clock valuation* is a mapping $\xi : X \mapsto \mathbb{R}_{\geq 0}$ that assigns to each clock a positive real value. We denote by $\mathbb{R}^X_{\geq 0}$ the set of clock valuations overt the set of clocks X. We say that a clock valuation ξ satisfies a guard g, denoted by $\xi \models g$, when the formula obtained by replacing in g each clock x by its valuation $\xi(x)$ is valid. For a clock valuation ξ and a subset of clocks $Y \subseteq X$, we denote by $\xi[Y]$ the clock valuation verifying that $\xi[Y](x) = 0$ if $x \in Y$ and $\xi[Y](x) = \xi(x)$ otherwise. We can now define Timed Broadcast Protocols which extend Broadcast Protocols with clocks.

Definition 7.4 (Timed Broadcast Protocol)**.** *A* Timed Broadcast Protocol *BP is a tuple* $(Q, X, M, \Delta, q_{in})$ *where:*

- Q *is a finite set of control states,*
- X *is a finite set of clocks,*
- M *is a finite alphabet of messages,*
- $\Delta \subseteq Q \times \mathtt{G}(X) \times \mathtt{BAct}(M) \times 2^X \times Q$ *is a finite set of edges such that each edge is labelled by a guard, a broadcast action and a subset of clocks to reset, and,*
- $q_{in} \in Q$ *is the initial contro state.*

Figure 7.10 provides an example of a Timed Broadcast Protocol with one clock x and three types of messages that can be broadcasted ($m1$, $m2$ and $m3$).

We fix now a Timed Broadcast Protocol $BP = (Q, X, M, \Delta, q_{in})$. The configurations of the Timed Ad Hoc Networks associated to BP are $(Q \times \mathbb{R}^X_{\geq 0})$-graphs, there interpretation is the same as for Ad Hoc Networks, with the difference that each node carries as well a clock valuation. We denote by Θ this set of configurations. Instead of writing a configuration

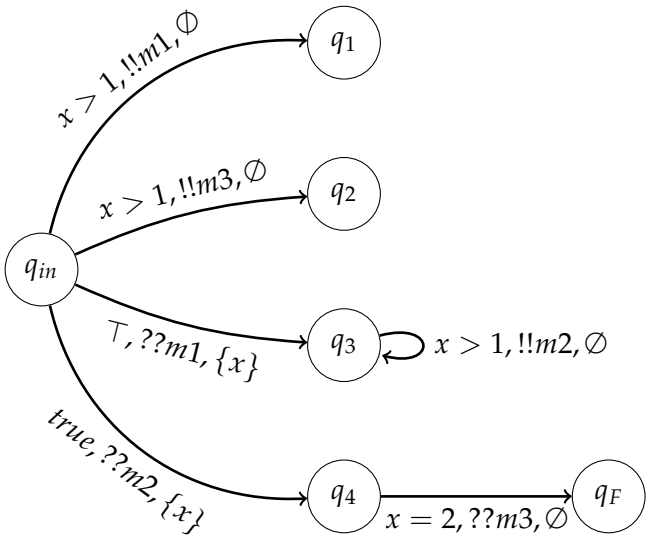

Figure 7.10: A Timed Broadcast Protocol

$\theta = (V, E, L)$ with $L : V \mapsto Q \times \mathbb{R}^X_{\geq 0}$, we will write it $\theta = (V, E, L, \xi)$ with $L : V \mapsto Q$ and $\xi : V \mapsto \mathbb{R}^X_{\geq 0}$ to simplify some notations and we denote by $L(\theta)$ the set $\{L(v) \mid v \in V\}$. A configuration $\theta = (V, E, L, \xi)$ is initial if $L(v) = q_{in}$ and $\xi(v)(x) = 0$ for all $v \in V$ and $x \in X$. The set of initial configurations is denoted by Θ_{in}.

As for Ad Hoc Networks, before to provide the semantics associated to Timed Broadcast Protocol, we need to express which node can (and will) receive a broadcast message. We consider a configuration $\theta = (V, E, L, \xi)$, a node $v \in V$ and a message $m \in \Sigma$, we use the same notations as for Ad Hoc Networks and define the set $R^\theta_m(v) = \{v' \in V \mid (v, v') \in E$ and $\exists (q, g, ??m, Z, q') \in \Delta$ s.t $L(v') = q$ and $\xi(v') \models g\}$ characterizing the neighbors of $v \in \theta$ which can receive the message m. Note that here the clock valuation associated to the node should as well satisfy the guard on the transition performing the reception of the message m. The timed transition system $TAHN(BP)$ induced by BP is then given by the tuple $(\Theta, \rightarrow)$ where $\rightarrow \subseteq \Theta \times (\Delta \cup \mathbb{R}_{\geq 0}) \times \Theta$ is the timed transition relation which we shall now define. We first describe the discrete transitions: for $\delta \in \Delta$, $\theta \xrightarrow{\delta} \theta'$ if and only if $\theta = (V, E, L, \xi)$ and $\theta' = (V, E, L', \xi')$ and $\delta = (q, g, a, Z, q')$ and one the following conditions holds:

- **Internal action:** $a = \tau$ and there exists $v \in V$ such that $L(v) = q$, $\xi(v) \models g$, $L'(v) = q'$ and $\xi'(v) = \xi(v)[Z]$ and for all $v' \in V \setminus \{v\}$, we have $L(v') = L'(v')$ and $\xi'(v') = \xi(v')$;
- **Broadcast communication:** $a = !!m$ and there exists $v \in V$ such that $L(v) = q$, $\xi(v) \models g$, $L'(v) = q'$ and $\xi'(v) = \xi(v)[Z]$ and for all $v' \in R^\theta_m(v)$, there exists an edge $(L(v'), g', ??m, Z', L'(v')) \in \Delta$ such that $\xi(v') \models g'$ and $\xi'(v') = \xi(v')[Z']$ and for all $v' \in V \setminus R^\gamma_m(v)$, we have $L'(v') = L(v)$ and $\xi'(v') = \xi(v')$.

For the timed transition: for $d \in \mathbb{R}_{\geq 0}$, we have $\theta \xrightarrow{d} \theta''$ if and only if:

- **Time:** $\theta = (V, E, L, \xi)$ and $\theta' = (V, E, L, \xi')$ and $\xi'(v)(x) = \xi(v)(x) + d$ for all $v \in V$ and $x \in X$.

The main change with respect to the semantics of Ad Hoc Networks is that we take into account the guards and the reset on clocks and as well that we have now some transitions which let time pass. We denote by $\theta \to \theta'$ if there exists $\delta \in \Delta$ such that $\theta \xrightarrow{\delta} \theta'$ or $d \in \mathbb{R}_{\geq 0}$ such that $\theta \xrightarrow{d} \theta'$. Finally, $\to^*$ represents the reflexive and transitive closure of $\to$.

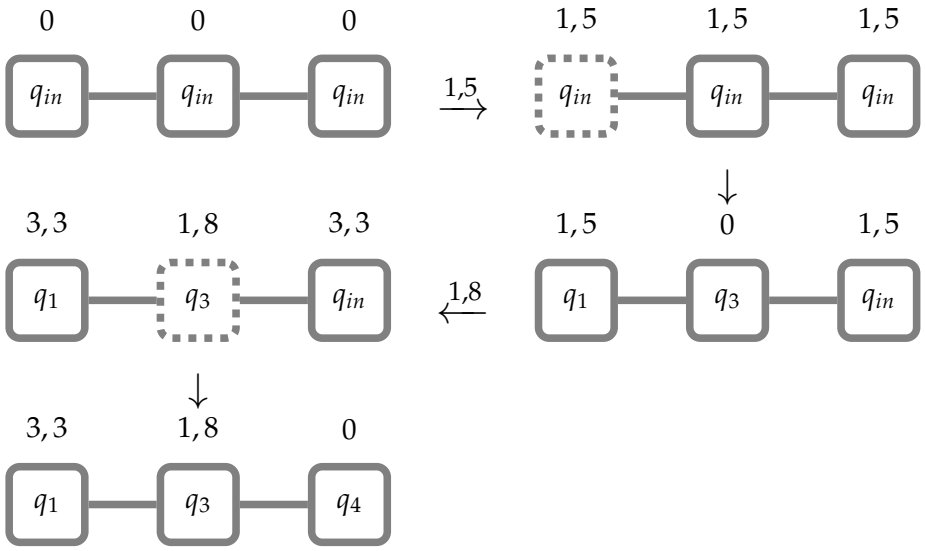

Figure 7.11: A finite run in the Timed Ad Hoc Network associated to the protocol of Figure 7.10

Example 7.5. *On Figure 7.11, we have depicted a run in the Timed Ad Hoc Network depicted on Figure 7.10. In each configuration, we have indicated the node which was responsible of the broadcast and we have indicated the delay above the time transition. For instance, the first transition lets time elapse for* $1,5$ *time units and afterwards the left node can broadcast the message* $m1$ *received by the center node which moves to state* q_3 *and resets its clock.*

The problem we study in this section is the pendant of AHN-ControlReach adapted to Timed Ad Hoc Networks and can be stated as follows.

TAHN-ControlReach	
Input:	A Timed Broadcast Protocol $BP = (Q, X, M, \Delta, q_{in})$ and a control state $q_f \in Q$;
Question:	Does there exists an initial configuration θ_{in} and a configuration θ such that $q_f \in L(\theta)$ and $\theta_{in} \to^* \theta$ in $TAHN(BP)$?

7.4.2 *Undecidability results*

First as Timed Ad Hoc Networks are an extension of Ad Hoc Networks, using Theorem 7.1, we know that in its full generality TAHN-ControlReach is undecidable. We first show that the decidability results obtained in the case of Ad Hoc Networks, see Theorems 7.2 and 7.4, cannot be extended to the case of Timed Ad Hoc Networks. To obtain these

new undecidability results, we use a reduction towards a similar problem as TAHN-ControlReach but in a different context. In [AJ03] , the authors study indeed a similar problem for networks of entities equipped with clocks, the main difference being that the communication is not performed thanks to broadcast but by pairwise rendez-vous. The introduced model is called Timed Network and has the following feature:

- a Timed Network contains a distinguished controller that is a finite state system without any clock;
- Each process in a Timed Network may communicate with all the other processes and hence there is no need for an underlying communication topology;
- Communication takes place through rendez-vous between fixed set of processes (all the processes involved in the rendez-vous changing their state according to the defined rules).

They show that if in Timed Networks, nodes are equipped with two clocks, then the parameterised control state reachability problem which asks whether there exists an initial configuration from which it is possible to reach a configuration exhibiting a given control state is undecidable. However, this problem becomes decidable if each node has a single clock.

First, we show that we can reduce the undecidability result for Timed Networks to Timed Ad Hoc Networks when the considered configurations are complete. As for Ad Hoc Networks, we say that a configuration $\theta = (V, E, L, \xi)$ of a Timed Ad Hoc Network is *complete* if, and only if, $E = V \times V \setminus \{(v, v) \mid v \in V. \ \}$. We remark indeed that if the considered configurations in Timed Ad Hoc Networks are complete, then it is easy to extract a controller (the first process to emit a broadcast message received by all the others becoming the controller) and as well to simulate rendez-vous communications (for instance if it is a pairwise rendez-vous, a node sends a message to initiate the rendez-vous, this message is received by all the other nodes, and the first one to answer, acknowledges the rendez-vous to the sender and at the same time puts all the other nodes in the state they were before the rendez-vous to be initiated). This leads us to the following undecidability result.

Theorem 7.5. *[Abd+11; Abd+16b]* TAHN-ControlReach *restricted to complete configurations and to Timed Broadcast Protocols with two clocks is undecidable.*

For what concerns configurations with bounded paths, we also obtain an undecidability result. For this we consider configurations whose topology forms what we call a star. We say that a configuration $\theta = (V, E, L, \xi)$ of a Timed Ad Hoc Network is a *star* of radius ℓ with $\ell \geq 1$ if and only if there is a partition of V of the form $\{v_0\} \cup V_1 \cup \ldots \cup V_\ell$ such that the following conditions are satisfied:

- $(v_0, v_1) \in E$ for all $v_1 \in V_1$, and,
- for each $i \in [1, \ell - 1]$ and $v_i \in V_i$ there is a unique $v_{i+1} \in V_{i+1}$ such that $(v_i, v_{i+1}) \in E$ and there is a unique $v_{i-1} \in V_{i-1}$ such $(v_i, v_{i-1}) \in E$, and,
- E does not contains other edges.

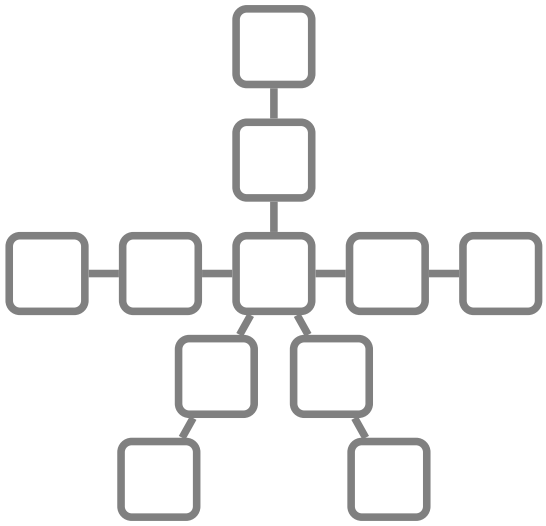

Figure 7.12: A radius 2 star

Figure 7.7 shows an example of a star of radius 1 (where the values of clocks are omitted) and Figure 7.12 an example of star of radius 2 (where the labels and the clocks values are omitted). In [Abd+11; Abd+16b], we have shown how to simulate a Timed Network working with two clocks per node with a Timed Ad Hoc Network with a single clock per process and taking into account only radius 2 stars, as allowed configurations. The idea is that the central node of the star simulates the controller of the Timed Network (and as well the rendez-vous rules) and each ray of the star which includes two nodes equipped with a single clock simulates a node of the Timed Network. First note, that it is possible to extract the controller in such a Timed Ad Hoc Network as the central node of the star is the only one that is connected to strictly more than 3 nodes, hence it suffices to broadcast a message and if a node receives three acknowledgment of it, then we know it is the central node. The main difficulty lies in the simulation of the rendez-vous transitions, since in the considered Timed Ad Hoc Network each node has access to a single clock. However the simulation is made possible thanks to the fact that in Timed Networks (as in Timed Ad Hoc Networks) clocks are not compared between each other, it is hence possible to simulate a guard over two clocks using the clocks of two processes in the same ray of the star configurations. This simulation leads us to state this other undecidability result.

Theorem 7.6. *[Abd+11; Abd+16b]* TAHN-ControlReach *restricted to stars of radius* 2 *configurations and to Timed Broadcast Protocols with one clock is undecidable.*

7.4.3 *Decidability results*

We will now see in which matter the two undecidability results we have just presented are tight. First, following the same path as the method proposed in [AJ03] to show that the parameterised reachability of a control state in Timed Networks where processes have a single clock is decidable, we prove that if we consider Timed Broadcast Protocols with a single clock then TAHN-ControlReach restricted to complete configurations is decidable. To obtain this result, we rely again on the theory of Well-Structured Transition Systems, but

we cannot find directly an order over configurations of Timed Ad Hoc Networks (mostly because of the clocks which are interpreted over the real) and hence the trick consists in representing such configurations differently. Our proof is based on the following step:

1. Define a symbolic way to represent set of configurations.
2. Exhibit a well-quasi-order over symbolic configurations.
3. Show that it is possible to compute symbolically the predecessors of a symbolic configuration.
4. Give an iterative method to compute all the elements from which a given symbolic configuration can be reached. Termination is ensured by the well-quasi-order over symbolic configurations.

We will now present the symbolic configurations our reasoning is based on. This symbolic representation is very similar to the one proposed in [AJ03] to represent configurations of Timed Networks, the main difference being that in Timed Ad Hoc Networks we do not have to deal with a controller and the computation of the discrete symbolic predecessor configurations is different since we consider broadcast and not rendez-vous communication. We fix a Timed Broadcast Protocol $BP = (Q, \{x\}, M, \Delta, q_{in})$ with a single clock x and we denote by $\mathtt{max}$ the maximal constant occurring in the guards of BP. A symbolic configuration η for BP is a tuple $(m, L^{symb}, \xi^{symb}, \sqsubseteq)$ such that:

- $m \in \mathbb{N}$ so that $[1, m]$ is the set of indices for the processes present in the network (recall that since these symbolic configurations are used to represent complete configurations, we do not need to consider a communication topology but only a set of processes),
- $L^{symb} : [1, m] \mapsto Q$ assigns to each process a state of BP,
- $\xi^{symb} : [1, m] \mapsto [0, \mathtt{max}]$ assigns to each process a natural less or equal than $\mathtt{max}$ corresponding to the integral part (up to $\mathtt{max}$) of the clock of the process;
- $\sqsubseteq$ is a total preorder on the set $[1, m] \cup \{\top, \bot\}$ verifying the following conditions,
  - $\bot$ and $\top$ are respectively the minimal and maximal elements of $\sqsubseteq$ with $\bot \neq \top$,
  - for $j \in [1, m]$, if $\xi^{symb}(j) = \mathtt{max}$ then $j \equiv \top$or $j \equiv \bot$ (where $a \equiv b$ iff $a \sqsubseteq b$ and $b \sqsubseteq a$),
  - for $j \in [1, m]$, if $j \equiv \top$ then $\xi^{symb}(j) = \mathtt{max}$.

The relation $\sqsubseteq$ provides an ordering for the process corresponding to the ordering of the fractional parts of their respective clock value and if $j \equiv \bot$ it means that the clock value of process j is at most $\mathtt{max}$ and its fractional part is 0, and if $j \equiv \top$ then the clock value of process j is strictly bigger than $\mathtt{max}$.

Remark. *$(\xi^{symb}, \sqsubseteq)$ corresponds exactly to the clock regions for the m clocks represented in the symbolic configuration η. This region construction was originally introduced in [AD94] for the analysis of timed automata since it allows to get rid of the precise values of the clocks by keeping an abstraction over the possible different values.*

In order to explain fully, the interpretation behind such symbolic configurations, we shall now define the set of configurations $[\![\eta]\!]$ representing by one such symbolic configuration η. Let $\eta = (m, L^{symb}, \xi^{symb}, \sqsubseteq)$ be a symbolic configuration and $\theta = (V, E, L, \xi)$ be a complete configuration. We have $\theta \in [\![\eta]\!]$ if and only if there exists an injective function $h : [1, m] \mapsto V$ such that for all $i, j \in [1, m]$, we have:

- $L(h(i)) = L^{symb}(i)$, and,
- $min(\texttt{max}, \lfloor \xi(h(i)) \rfloor) = \xi^{symb}(i)$ (where $\lfloor a \rfloor$) denotes the integral part of a), and,
- $i \equiv \bot$ if and only if $\xi(h(i)) \leq \texttt{max}$ and $frac(\xi(h(i))) = 0$ (where $frac(a)$ denotes the fractional part of a),
- $i \equiv \top$ if and only if $\xi(h(j)) > \texttt{max}$,
- if $i \not\equiv \bot$ and $j \not\equiv \top$ then $frac(\xi(h(i))) \leq frac(\xi(h(j)))$ if, and only if $i \sqsubseteq j$.

Note that in the above definition, we do not require the number of nodes in θ and the number of processes in η to be the same, but only that each process in η can be matched with a process in θ. We denote by H_{symb} the set of symbolic configurations and we consider the quasi-order $(H_{symb}, \preceq)$ defined as follows: given two symbolic configurations $\eta_1 = (m_1, L_1^{symb}, \xi_1^{symb}, \sqsubseteq_1)$ and $\eta_2 = (m_2, L_2^{symb}, \xi_2^{symb}, \sqsubseteq_2)$ in H^{symb}, we have $\eta_1 \preceq \eta_2$ if and only if there exists an injective mapping $g : [1, m_1] \mapsto [1, m_2]$ such that for all $i, j \in [1, m_1]$ the following conditions hold:

- $L_2^{symb}(g(i)) = L_1^{symb}(i)$,
- $\xi_2^{symb}(g(i)) = \xi_1^{symb}(i)$,
- $g(i) \equiv_2 \bot$ if and only if $i \equiv_1 \bot$ (where for $k \in \{1, 2\}$, $a \equiv_k b$ if and only if $a \sqsubseteq_k b$ and $b \sqsubseteq_k a$),
- $g(i) \equiv_2 \top$ if and only if $i \equiv_1 \top$,
- $g(i) \sqsubseteq_2 g(j)$ if and only if $i \sqsubseteq_1 j$.

We then prove in [Abd+11; Abd+16b] that given $\eta_1, \eta_2 \in H_{symb}$ we have that $\eta_1 \preceq \eta_2$ if and only if $[\![\eta_2]\!] \subseteq [\![\eta_1]\!]$ and that $(H^{symb}, \preceq)$ is a well-quasi-order.

The last part of our reasoning consists in showing that given a symbolic configuration η, we can compute a set of symbolic configurations $\{\eta_1, \ldots, \eta_k\}$ such that for each $\theta \in [\![\eta]\!]$ and θ' verifying $\theta' \to \theta$ in $TAHN(BP)$, there exists $\eta' \in \{\eta_1, \ldots, \eta_k\}$ such that $\theta' \in [\![\eta']\!]$. To conclude, we prove thanks to the well-quasi-order $(H_{symb}, \preceq)$ that the iterative computation of the symbolic predecessors terminate.

A similar method based on symbolic configurations can be used if the considered configurations are stars of radius 1 (and the Timed Broadcast Protocol has again a single clock), the main differences being that in the symbolic configurations we have to take care of an extra process corresponding to the center of the stars and that the computation of the predecessors is a bit different. This allows us to state the following decidability results in the case of Timed Ad Hoc Network.

Theorem 7.7. *[Abd+11; Abd+16b]*

1. TAHN-ControlReach *restricted to complete configurations and to Timed Broadcast Protocols with one clock is decidable.*
2. TAHN-ControlReach *restricted to stars of radius* 1 *configurations and to Timed Broadcast Protocols with one clock is decidable.*

8

VERIFICATION OF RECONFIGURABLE BROADCAST NETWORKS

When we introduced the model of Ad Hoc Networks in [DSZ10], we showed that apart from restricting the considered configurations (see Theorems 7.2 and 7.4), there exists another way to regain decidability for the parameterised control state reachability problem by allowing non-deterministic reconfiguration of the communication topology. We studied this new semantic not only to obtain decidability results but as well because it can be interpreted as mobility of the entities in Ad Hoc Networks.

In this new model, we will refer to as Reconfigurable Broadcast Network, as in Ad Hoc Network, all entities follows the same Broadcast Protocol and the communication is achieved as well thanks via a node which broadcast a message to its neighbours, but at any moment, the communication topology can change arbitrarily. This reconfiguration can be seen, from an abstract point of view as mobility of the entities inside the networks: when an edge of the topology is deleted, it means that the two concerned nodes have moved away one from each other and when a new edge appears, it means that the nodes have come close one to each other. Of course, since we do not put any constraints on the new communication topology obtained after a reconfiguration step, it does not correspond directly to a real mobility where one might want to take into account more geometrical constraints in order for instance to represent the fact that when a node moves, it has to follow a certain path at a certain speed and as a matter of fact from a given communication topology, not all topologies can be reached during a reconfiguration. However building a model which allows this kind of geometrical consideration is a hard task itself and in our works we choose to study a simpler version of this mobility for the following reasons: it eases the definition of the model and the verification process and it is not completely meaningless as our semantics can correspond to an overapproximation of a more constrained one. Indeed, if no bug is found in the model where reconfiguration can be performed in a complete uncontrolled fashion, then no bug will occur in the model where reconfiguration is more restricted.

Introducing reconfiguration in Ad Hoc Networks consists in relaxing the reachability transition of these networks in a sense that it allows more behaviours of the model. There exist other works in the field of verification of infinite state systems, where such relaxations of the relation transition have been studied and for which the verification process has become easier than in the original model. This is in particular the case for Lossy Channel Systems [AJ96]: these systems correspond to a finite number of finite state machines communicating thanks to FIFO channel. When the channels are reliable, i.e. no message is lost, then simple verification problems such as the reachability of a control state are undecidable, whereas if the channels are unreliable, then this latter problem becomes decidable (but non primitive recursive [Sch02]). Similarly, in [May03], the author considers a model of counter machines with increment, decrement and zero tests and where at any time the counter can be decreased non-deterministically and he shows that here again this relaxation allows to obtain the decidability of the reachability of a control state (which is undecidable without the relaxation, see Theorem 3.1). In [DL09], the authors have considered a similar model but they allow incrementing errors instead of decrementing errors and here again the relaxation allows to jump the gap from undecidability to decidability for some verification problems.

Note that the reconfiguration we introduce in the Ad Hoc Networks semantics could as well be seen as an unreliable communication, i.e. the broadcasted message can be lost, indeed to simulate the lost of a message between two entities, the reconfiguration can detach the receiver from the emitter just before the message is sent and right after bring back the receiver. We shall see however that, surprisingly, in Reconfigurable Broadcast Networks the verification process has not a high complexity as it is the case for lossy channel systems or counter systems with increasing errors.

CONTRIBUTIONS. The first time we introduced Reconfigurable Broadcast Networks was in [DSZ10] where we showed that for this model the parameterised reachability of a control state is decidable and in EXPSPACE thanks to a reduction towards the control state reachability problem in VASS (see Theorem 3.5). However we observed later on in [Del+12] that our verification process could be improved and hence we obtained tight complexity bounds for various reachability questions where we allow to count the number of processes in ech state in the configuration to be reached.

Encouraged by theses decidability results we obtained (together with the nice complexity bounds), we decided to study other problems on Reconfigurable Broadcast Networks and their possible extensions. In [BFS15], we studied reachability problems for Reconfigurable Broadcast Networks adding a restriction on the considered executions, indeed we required that all the processes in the networks must follow the same local strategy. With this last term, we mean that all processes with the same sequence of last performed actions and received messages do the same action. This restriction reflects in a sense the distributed aspect of our networks where we would like that each process behaves the same way with the same information. This problem can be seen as a synthesis problem where we want to extract a deterministic behaviour for the different processes of our networks from a non-deterministic broadcast protocol.

We have then studied two extensions of Reconfigurable Broadcast Networks. In [BFS14], we studied a probabilistic version of these networks where each process can perform an internal action probabilistically, i.e. from some states there is a probabilistic distribution over the set of states a process can move to. This allows to model a randomised change of the internal state of a process (this change could be caused by an external component whose probability is known). The semantics of this model is then given in terms of a Markov Decision Process with an infinite number of states over which we solved qualitative reachability questions: as for instance, does there exists a number of entities which allows to reach almost surely a control state of the protocol. The technique we used to solve these latter problems is particularly interesting as it is based on a reduction towards a two player game played on the graph of a Reconfigurable Broadcast Network with parity objective.

In [DST13; DST16], we have extended the message alphabet of Reconfigurable Broadcast Networks to data from an infinite set. In order to manipulate these data in a finite representation, we equipped each process with a finite set of registers where the received data can be stored and we allowed a process to test for equality between a received data and a stored one. We have here again studied some parameterised reachability questions on this model and investigated the impact on decidability and complexity of different restrictions as for instance the number of registers or the number of values carried by each single message.

8.1 REACHABILITY IN RECONFIGURABLE BROADCAST NETWORKS

In this section, we present the results we obtained for the verification of reachability properties in Reconfigurable Broadcast Networks. As for Ad Hoc Networks, we assume in these networks, that each entity or process executes the same protocol given by a Broadcast Protocol (see Section 7.1.1) and that a during an execution the size of the network does not evolve. The main difference being that at each step the communication topology might change in a complete non-deterministic way.

8.1.1 *Reconfigurable Broadcast Networks induced by a Broadcast Protocol*

We first introduced Reconfigurable Broadcast Networks in [DSZ10] by extending the semantics of Ad Hoc Networks with a reconfiguration step which could happen at any time and whose role is to change the communication topology of the network without altering the number of entities, basically a reconfiguration step changes the edges of a configuration (which is a graph, see Section 7.1.2), but not the set of vertices. However, to deal with the verification questions we are interested in, we can abstract this reconfiguration step and delete away the graph structure of the configurations and keep only track on the number of processes in a certain state of the protocol. In this abstraction, to respect the semantics of reconfigurable networks, we assume that when a broadcast is performed, only a non-deterministically chosen set of entities will receive the emitted message, this set corresponding to the entities connected to the emitter at the moment of the broadcast. We present here this semantics where the reconfiguration step is abstracted away and substituted by a broadcast to some of the entities in the network.

We are now ready to move to the definition of a Reconfigurable Broadcast Network induced by a Broadcast Protocol $BP = (Q, M, \Delta, q_{in})$. A configuration of such a network is given by a vector $\lambda \in \bigcup_{\ell \in \mathbb{N}\setminus\{0\}} Q^\ell$. For a configuration $\lambda \in Q^\ell$ with $\ell \geq 1$, we denote by $\|\lambda\| = \ell$ its dimension, intuitively it corresponds to the number of entities/processes present in the network and $\lambda[p]$ represents the state of the p-th process for each $1 \leq p \leq \|\lambda\|$. The set of configurations of the Reconfigurable Broadcast Network induced by BP is then $\Lambda = \bigcup_{\ell \in \mathbb{N}\setminus\{0\}} Q^\ell$ and the set of initial configurations is $\Lambda_{in} = \bigcup_{\ell \in \mathbb{N}\setminus\{0\}} \{q_{in}\}^\ell$. Hence in an initial configuration, all processes are in state q_{in}.

The transition system $RBN(BP)$ induced by BP is then given by the tuple $(\Lambda, \Rightarrow)$ where $\Rightarrow \subseteq \Lambda \times \mathbb{N} \times \Delta \times 2^{\mathbb{N}} \times \Gamma$ is the transition relation which we shall now define. We have $\lambda \xRightarrow{p,\delta,R} \lambda'$ if and only if $\|\lambda\| = \|\lambda'\|$ and $p \in [1, \|\lambda\|]$ and $R \subseteq [1, \|\lambda\|] \setminus \{p\}$ and one of the following conditions holds:

- **Internal action:** $\delta = (\lambda[p], \tau, \lambda'[p])$ and $R = \emptyset$ and $\lambda'[p'] = \lambda[p']$ for all $p' \in [1, \|\lambda\|] \setminus \{p\}$;
- **Broadcast communication:** $\delta = (\lambda[p], !!m, \lambda'[p])$ and $(\lambda[p'], ??m, \lambda'[p']) \in \Delta$ for all $p' \in R$ such that $\{(\lambda[p'], ??m, q) \in \Delta \mid q \in Q\} \neq \emptyset$ and $\lambda'[p''] = \lambda[p'']$ for all $p'' \in [1, \|\lambda\|] \setminus (R \cup \{p\})$ and for all $p'' \in R$ such that $\{(\lambda[p'], ??m, q) \in \Delta \mid q \in Q\} - \emptyset$.

Note that here we label the transition relation with p, the process performing an internal action or a broadcast, δ the edge of the protocol taken by process p and R a reception set which determines which processes will receive the broadcast. For the next section, we do not really need in our definition p and R but introducing this term will make sense when

we will deal with local strategies. If we sum up the semantics, with an internal action, only the process p changes its state and when a process p broadcasts a message m then all the processes in R than can receive this message will receive it and change their state accordingly. Note that here again, since we require that $\|\lambda\| = \|\lambda'\|$, it means that the number of processes remains the same during an execution (there is no creation or deletion of processes). As for Ad Hoc Networks, we denote by $\lambda \Rightarrow \lambda'$ if there exists $p \in \mathbb{N}$, $\delta \in \Delta$ and $R \in 2^{\mathbb{N}}$ such that $\lambda \xRightarrow{p,\delta,R} \lambda'$ and $\Rightarrow^*$ represents the reflexive and transitive closure of $\Rightarrow$. Given an initial configuration $\lambda_{in} \in \Lambda_{in}$, a finite run (or finite execution) starting from λ_{in} in $RBN(BP)$ is a finite path in $RBN(BP)$ denoted as:

$$\rho := \lambda_0 \xRightarrow{p_0,\delta_0,R_0} \lambda_1 \xRightarrow{p_1,\delta_1,R_1} \dots \xRightarrow{p_{n-1},\delta_{n-1},R_{n-1}} \lambda_n$$

with $\lambda_0 \in \Lambda_{in}$. We then denote by $\|\rho\|$ the natural $\|\lambda_0\|$ corresponding to the number of processes involved in the execution ρ. Note that by definition $\|\rho\| = \|\lambda_0\| = \dots = \|\lambda_n\|$.

Example 8.1. *On Figure 8.2, we have represented a possible run in the Reconfigurable Broadcast Network associated to the Broadcast Protocol depicted on Figure 8.1. To see the fundamental difference with Ad Hoc Networks, note that in the first step, the process* 1 *emits a message* $m1$ *which is received only by process* 3 *(the number of each process being written in the first configuration) and not by process* 2*, otherwise this latter process would have to change its state from* q_{in} *to* q_3*, but in the second step process* 2 *emits a message which is received by process* 1*. It could be seen as a change in the communication topology between these two steps: in the second step* 1 *and* 2 *being adjacent but not in the first step. Furthermore in the second step, we remark that* 4 *belongs to the set of receivers for the message* $m2$ *emitted by* 2*, but since in* q_{in} *there is no outgoing transition labelled with* $??m2$*, process* 4 *does not change its state.*

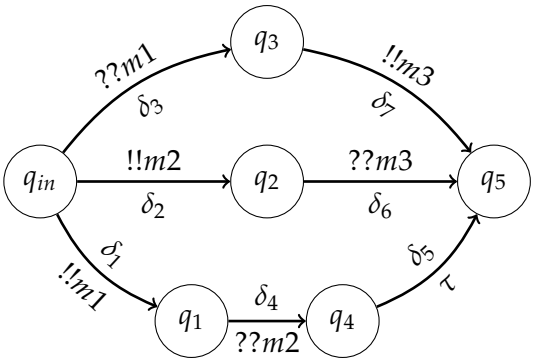

Figure 8.1: A Broadcast Protocol

8.1.2 *Reachability of counting constraints*

In [DSZ10], we have shown that the parameterised control state reachability problem and the parameterised target state reachability problem are both decidable when considering Reconfigurable Broadcast Networks instead of Ad Hoc Networks thanks to a reduction to the coverability and reachability problems in Petri nets (or equivalently control state reachability and reachability in Vector Addition with States). However, in [Del+12], we designed algorithms to solve these problems with a much better complexity. Actually to

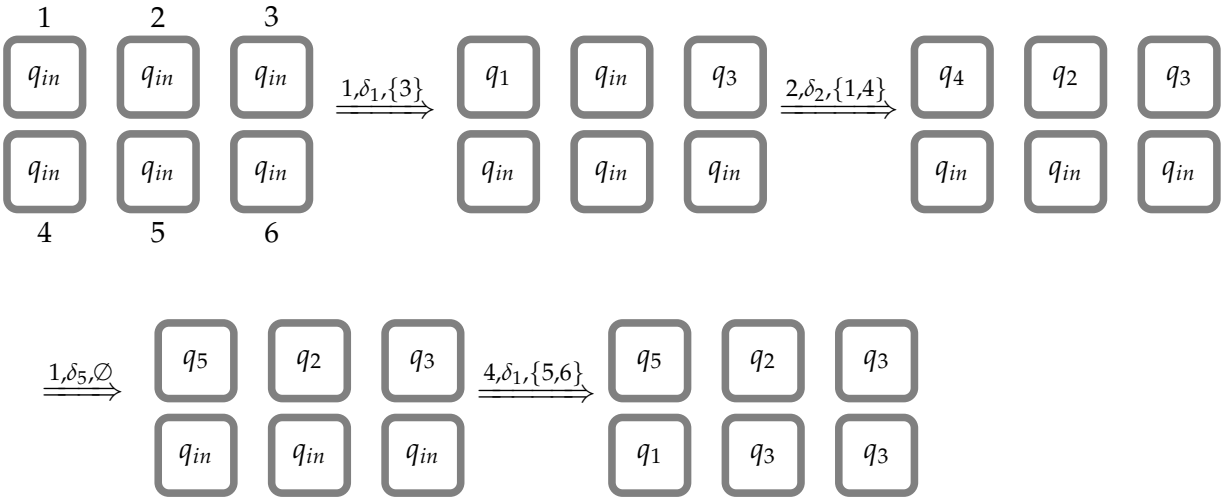

Figure 8.2: A run in the Reconfigurable Broadcast Network associated to the protocol of Figure 8.1

fully understand where lies the difficulty in solving reachability queries in Reconfigurable Broadcast Networks, we introduced a new problem which consists in checking whether a configuration satisfying a logical constraint on the number of present states can be reached. We then provide different algorithms to solve this problem depending on the shape of the counting constraint.

We first define formally these counting contratins and how we interpret them. Let $BP = (Q, M, \Delta, q_{in})$ be a broadcast protocol. The set of counting constraints for BP is given by the following grammar:

$$\phi ::= a \leq \#q < b \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg\phi$$

where $a \in \mathbb{N}$, $q \in Q$ and $b \in (\mathbb{N} \setminus \{0\}) \cup \{+\infty\}$. We denote by $\mathtt{CC}$ the class of counting constraints, by $\mathtt{CC}[\geq 1]$ the class in which negation is forbidden and atomic formulas have only the form $1 \leq \#q < +\infty$ and finally by $\mathtt{CC}[\geq 1, = 0]$ the class of counting constraints in which negation is forbidden and atomic formulas have only the form $1 \leq \#q < +\infty$ or $0 \leq \#q < 1$.

Intuitively, a counting constraint of the form $a \leq \#q < b$ specifies that in a considered configuration the number of processes which is in state q should be bigger that a and strictly smaller b. Given a configuration λ of $RBN(BP)$, for $q \in Q$, we use the notation $\#q(\lambda)$ to represent the value $|\{p \in \mathbb{N} \mid 1 \leq p \leq \|\lambda\| \text{ and } \lambda[p] = q\}|$, in other words it corresponds to the number of processes in λ being in the control state q. Given a counting constraint for BP and a configuration λ of $RBN(BP)$, we then define the satisfaction relation $\lambda \models \phi$ as follows: for atomic formula, we have $\lambda \models a \leq \#q < b$ if and only if $a \leq \#q(\lambda) < b$ and the cases of the boolean combinations directly follow.

We are now ready to present the counting reachability problem for Reconfigurable Broadcast Networks to which is dedicated this section.

RBN-CountingReach	
Input:	A Broadcast Protocol $BP = (Q, M, \Delta, q_{in})$ and a counting constraint ϕ for BP;
Question:	Does there exists an initial configuration λ_{in} and a configuration λ such that $\lambda_{in} \Rightarrow^* \lambda$ in $RBN(BP)$ and $\lambda \models \phi$?

8.1.3 *Solving the counting reachability problem*

In [Del+12], we have determined the precise complexity bounds for RBN-CountingReach taking into account whether the provided counting contraint belongs to $\mathtt{CC}[\geq 1]$, $\mathtt{CC}[\geq 1, = 0]$ or to the general class $\mathtt{CC}$.

8.1.3.1 *Counting constraints in* $\mathtt{CC}[\geq 1]$

If the counting constraint belongs to $\mathtt{CC}[\geq 1]$, it means that it can only test the presence of some control states in a configuration. It corresponds to a generalization of the parameterised control state reachability problem. We have shown that under this restriction RBN-CountingReach is Ptime-complete. To obtain the upper bound we observe the following property of Reconfiguration Broadcast Networks: if a configuration in which the control state q can be reached from an initial configuration λ_{in}, then there is another configuration than can be reached where q appears an arbitrary number of times from a larger initial configuration λ'_{in}. More specifically, λ'_{in} is obtained by replicating several times λ_{in} and we use then reconfiguration to mimic parallel executions of the execution witnessing originally the state q and reach a configuration with several repeated occurrences of q. Furthermore, since the considered constraint belongs to $\mathtt{CC}[\geq 1]$, we do not need to solve the counting reachability problem to count the occurrences of states and we just have to remember which states have been seen. As a consequence, we design a polynomial time procedure which at each step increases the set of seen states checking which edge of the protocol can be applied and add, whenever it is the case, new states to the considered set. For instance, if in this set there are two states q and q' and in the protocol two edges $(q, !!m, q'')$ and $(q', ??m, q''')$, we know that we can add states q'' and q''' and that there is a reachable configuration where each state q, q', q'' and q''' occurs an arbitrarily number of times. This procedure halts whenever the set of reachable states cannot be increased anymore. To obtain the lower bound we provide a reduction logarithmic in space from the Circuit Value Problem which is known to be Ptime-complete[Lad75].

8.1.3.2 *Counting constraints in* $\mathtt{CC}[\geq 1, = 0]$

If the counting constraint belongs to $\mathtt{CC}[\geq 1, = 0]$, it means that it can only test the presence or absence of some control states in a configuration and here again it is not necessary to count the number of occurrences of each state. This restriction corresponds to an extension of the parameterised target reachability problem (where we seek for a configuration where all the processes are in the same given state). We have shown that under this restriction RBN-CountingReach is NP-complete. As for the previous case, the decision procedure manipulates a set of seen control state and is based on the following observation. If there is

an execution of the Reconfigurable Broadcast Network which allows to reach a configuration λ satisfying a counting constraint $\phi \in \mathtt{CC}[\geq 1, = 0]$ then there is an execution which allows to reach another configuration λ' satisfying as well ϕ and this second execution can be divided into two consecutive phases verifying the following property: during the first phase the set of control states appearing in each configuration strictly increases and during the second phase it strictly decreases. The non-deterministic procedure consists then in guessing two sequences of set of control states, one that strictly increases and the other one that strictly decreases and then in checking that they correspond to an execution of the Reconfigurable Brodcast Network. By definition, these two sequences are of polynomial length in the number of states of the considered protocol. To obtain the lower bound we provide a reduction from the boolean satisfiability problem which is known to be NP-complete.

8.1.3.3 *Counting constraints in* CC

Finally, we proved that, in its full generality, RBN-CountingReach is Pspace-complete (assuming that the values provided in the counting constraint are encoded in unary). The main idea to solve the general case consists in relying on a symbolic representation of configurations in which the behavior of the network is observed precisely for a fixed number of nodes only, while for the other nodes only the set of present states is taken into account (and we forgot the number of occurrences).

We consider a broadcast protocol $BP = (Q, M, \Delta, q_{in})$ and a counting constraint ϕ for BP and we denote by $val(\phi)$ the largest natural constant that appears in ϕ. A symbolic configuration of $RBN(BP)$ and ϕ is then a pair (v, S) where $v \in Q^{|Q| \times val(\phi)}$ and $S \subseteq Q$. Intuitively v represents $|Q| \times val(\phi)$ processes that in the symbolic approach we simulate faithfully with respect to the semantics of Reconfigurable Broadcast Networks and in set S we store the states of other processes. Hence a symbolic configuration (v, S) represents all the configurations λ such that $\|\lambda\| \geq |Q| \times val(\phi)$ and $\#q(\lambda) > \#q(v)$ for all $q \in S$ and $\#q(\lambda) = \#q(v)$ for all $q \in Q \setminus S$. We then show how to compute the symbolic successors of a configuration and how we can test whether all the configurations of a symbolic configuration satisfy a counting constraint. This provides us with an algorithm which consists in looking in the graph of symbolic configurations where the transition relation is the symbolic successor configuration, whether it is possible to reach a symbolic configuration satisfying ϕ (ie such that all its configurations satisfy ϕ). Since the number of symbolic configurations is exponential and each symbolic configuration needs only a polynomial number of bits to be stored and since checking whether a symbolic configuration satisfies a counting constraint can be done in polynomial time so as checking whether a symbolic configuration is the symbolic successor of another one, we obtain a (non-deterministic) Pspace algorithm to solve RBN-CountingReach. To obtain the lower bound we provide a reduction from the reachability problem for 1-safe nets (Petri nets where in every reachable markings every place has at most one token) which is known to be Pspace-complete [CEP95].

8.1.3.4 *Summary of the results*

From what we have explained previously we hence have obtained the following complexity results for the counting reachability problem for Reconfigurable Broadcast Networks.

Theorem 8.1. *[Del+12]*

1. RBN-CountingReach *restricting to counting constraints in* $\mathtt{CC}[\geq 1]$ *is* Ptime*-complete.*
2. RBN-CountingReach *restricting to counting constraints in* $\mathtt{CC}[\geq 1, = 0]$ *is* NP*-complete.*
3. RBN-CountingReach *is* Pspace*-complete.*

Remark. *As already mentioned the parameterised target state reachability for Reconfigurable Broadcast Networks (defined as for Ad Hoc Networks but considering the semantics with reconfiguration) can be encoded in* RBN-CountingReach *with a constraint in* $\mathtt{CC}[\geq 1, = 0]$*, hence the previous theorem tells us that this problem is in* NP*. However this problem is* Ptime*-complete (as the parameterised control state reachability) as it is explained in the second chapter of [Fou15].*

8.2 VERIFICATION OF RECONFIGURABLE BROADCAST NETWORKS UNDER LOCAL STRATEGIES

In [BFS15], we have considered the parameterised reachability state and target state problems for Reconfigurable Broadcast Networks under a new perspective, which could have its interest for other models of networks where entities all follow the same protocol. Indeed in such models, the executed protocol is often non-deterministic in the sense that in a certain configuration, an entity in a certain state could have different possible options (for instance it can go to two different states while broadcasting or sending a message m). Therefore it may happen that two entities behave differently, even if they have the same information on what has happened so for in the execution. To forbid such non-truly distributed behaviours, in [BFS15], we constrain processes to take the same decision in case they have taken the same sequence of transitions in their past. We thus study the aforementioned reachability problems restricted to what we call local strategies. Interestingly, the notably difficult distributed controller synthesis problem [PR90] is relatively close to the problem of existence of a local strategy. Indeed a local strategy corresponds to a local controller for the processes executing the protocol and whose role is to resolve the non-deterministic choices.

8.2.1 *Local strategies*

We will present here the definition of local strategies for Reconfigurable Broadcast Networks which will ensure that in an execution all processes perform the same choices of edges according to their past history (ie the edges of the protocol it has fired so far).

We first need some preliminary definitions. We consider a Broadcast Protocol $BP = (Q, M, \Delta, q_{in})$. A finite path in BP is either the empty path, denoted by ϵ, or a non-empty finite sequence of consecutive edges $\delta_0 \cdots \delta_\ell$ with $\delta_i = (q_i, a, q_{i+1}) \in \Delta$ for all $i \in [0, \ell]$) and $q_0 = q_{in}$. We use the notation $last(\pi)$ to represent the last state of the path π: for the empty path, we will have $last(\epsilon) = q_{in}$ and if $\pi = \delta_0 \cdots \delta_\ell$ with $\delta_i = (q_i, a, q_{i+1})$, we set $last(\pi) = q_{\ell+1}$. We denote by $\mathtt{Path}(BP)$ the set of finite paths of BP.

A *local strategy* σ for BP is then a pair (σ_a, σ_r) of partial functions such that, given an history, σ_a specifies the next active edge to be taken (labelled by a broadcast or an internal

action) and σ_r the reception edge (labelled by a reception) to be chosen when receiving a message. Formally, we have:

- $\sigma_a : \texttt{Path}(BP) \mapsto (Q \times (\{!!m \mid m \in M\} \cup \{\tau\}) \times Q)$ is a partial function such that if $\sigma_a(\pi) = (q, a, q')$ for the finite path $\pi \in \texttt{Path}(BP)$ then $(q, a, q') \in \Delta$ and $q = last(\pi)$,
- $\sigma_r : \texttt{Path}(BP) \times M \mapsto (Q \times \{??m \mid m \in M\} \times Q)$ is a partial function such that if $\sigma_a(\pi, m) = (q, a, q')$ for the finite path $\pi \in \texttt{Path}(BP)$ and the message $m \in M$ then $(q, a, q') \in \Delta$ and $q = last(\pi)$ and $a = ??m$.

Since our aim is to analyze executions of Reconfigurable Broadcast Networks where each process behaves according to the same local strategy, we now need to provide the definition of such executions. We consider an execution $\rho := \lambda_0 \xrightarrow{p_0,\delta_0,R_0} \lambda_1 \xrightarrow{p_1,\delta_1,R_1} \dots \xrightarrow{p_{n-1},\delta_{n-1},R_{n-1}} \lambda_n$ of $RBN(BP)$. For $p \in \|\rho\|$, we first define $\pi_p(\rho)$ as the past of process p in ρ. It corresponds to the sequence $\kappa_0 \dots \kappa_{n-1}$ where for each $i \in \{0, n-1\}$, we have: $\kappa_i = \delta_i$ if $p_i = p$, or $\kappa_i = (\lambda_i[p], ??m, \lambda_{i+1}[p])$ if $p \in R_i$ and $\delta_i = (q, !!m, q')$ and $(\lambda_i[p], ??m, \lambda_{i+1}[p]) \in \Delta$, or $\kappa_i = \epsilon$ in the other cases (where we assume that $\epsilon.\delta = \delta.\epsilon = \delta$ for all $\delta \in \Delta$). Note that by definition we have $\pi_p(\rho) \in \texttt{Path}(BP)$. We then say that a finite path $\delta_0 \cdots \delta_\ell$ of BP respects a local strategy $\sigma = (\sigma_a, \sigma_r)$ if for all $i \in [0, \ell - 1]$, we have either $\delta_{i+1} = \sigma_a(\delta_0 \cdots \delta_i)$ or $\delta_{i+1} = \sigma_r(\delta_0 \cdots \delta_i, m)$ for some $m \in M$. Following this, an execution ρ respects a local strategy σ if and only if $\pi_p(\rho)$ respects σ for all $p \in [1, \|\rho\|]$ (i.e. each process participating in ρ behaves as specified by σ).

8.2.2 *Verification problems*

In [BFS15], we studied the parameterised control state and target state reachability problems for Reconfigurable Broadcast Networks restricted to local strategies. These two problems can be defined as follows:

RBN-Local-ControlReach	
Input:	A Broadcast Protocol $BP = (Q, M, \Delta, q_{in})$
	and a control state $q_f \in Q$;
Question:	Does there exist an execution $\rho := \lambda_{in} \Rightarrow^* \lambda$ in $RBN(BP)$
	and a local strategy σ
	such that ρ respects σ and $\#q_f(\lambda) > 0$?

RBN-Local-Target	
Input:	A Broadcast Protocol $BP = (Q, M, \Delta, q_{in})$
	and a control state $q_f \in Q$;
Question:	Does there exist an execution $\rho := \lambda_{in} \Rightarrow^* \lambda$ in $RBN(BP)$
	and a local strategy σ
	such that ρ respects σ and $\#q_f(\lambda) = \|\rho\|$?

As we have seen before, if we forget the fact that the considered executions have to respect some local strategies then these two problems are PTIME-complete and in [BFS15] we have shown that considering local strategies make them NP-complete.

Example 8.2. *To illustrate these verification problems, we provide an example of a Broadcast Protocol on Figure 8.3. On this protocol, there exists a local strategy which allows to reach the control state q_f. Consider indeed the following execution with two processes: $[q_{in}, q_{in}] \Rightarrow [q_1, q_{in}] \Rightarrow [q_f, q_1]$. It respects the local strategy which says that from an empty past, a process has to take the edge $(q_{in}, !!m, q_1)$ and in the state q_1 a process on reception of a message m has to take the edge $(q_1, ??m, q_f)$; in the first step of the execution, no process receives the broadcasted message and in the second step, only the first process receives it (we recall that in Reconfigurable Broadcast Networks at each step the set of processes that can receive a message possibly changes). On the other hand, no local strategy allows an execution which can reach the state q'_f. The reason being that to reach this state, one process with empty past needs to go to q_2 and another to q_1. Finally the execution $[q_{in}, q_{in}, q_{in}] \Rightarrow [q_1, q_{in}, q_3] \Rightarrow [q_1, q_1, q_4] \Rightarrow [q_t, q_t, q_t]$ brings the three processes in the target state q_t and it respects a local strategy.*

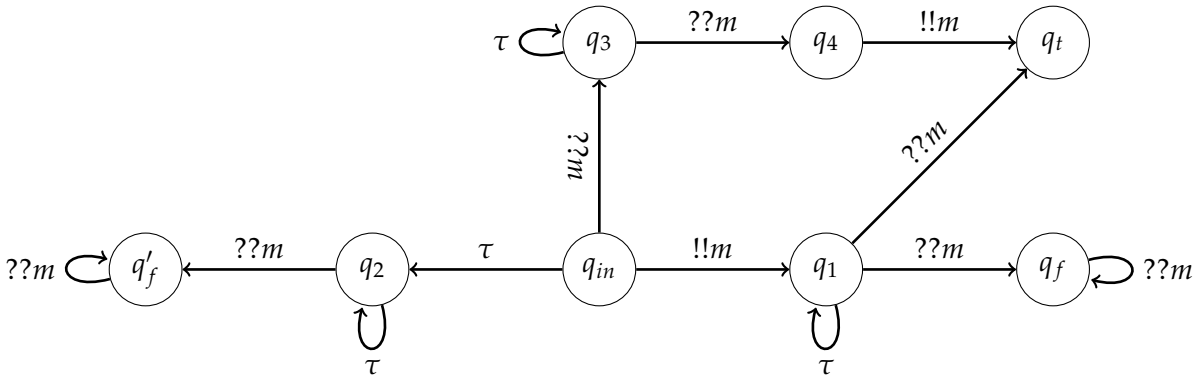

Figure 8.3: A Broadcast Protocol

8.2.3 *Solving reachability problems under local strategies*

In [BFS15], we have proved that RBN-LOCAL-CONTROLREACH and RBN-LOCAL-TARGET are NP-complete problems. To obtain this result, we use trees to represent local strategies and show that a control state can be reached in an execution respecting a local strategy if and only if there exists such a tree of polynomial size respecting some conditions. The idea behind these trees being that the paths in the tree represent past histories and the edges outgoing a specific node represent the decisions of the local strategy. We now present the different steps of this proof for RBN-LOCAL-CONTROLREACH.

We first provide the definition of our tree representation of strategies. A *strategy pattern* for a broadcast protocol $BP = (Q, M, \Delta, q_{in})$ is a labelled tree $T = (N, n_0, E, \Delta, lab)$ where N is a finite set of nodes, $n_0 \in N$ is the root of the tree, $E \subseteq N \times N$ is the edge relation and $lab : E \mapsto \Delta$ the edge labelling function. Moreover T is such that if $e_1 \dots e_\ell$ is a path in T (starting at n_0) then $lab(e_1) \dots lab(e_\ell)$ belongs to $\texttt{Path}(BP)$ and for every node $n \in N$ there is a most one edge $e = (n, n')$ such that $lab(e) \in Q \times (\{!!m \mid m \in M\} \cup \{\tau\}) \times Q$ and, for each message $m \in M$ there is at most one edge $e' = (n, n'')$ such that $lab(e') \in Q \times \{??m\} \times Q$.

Since all labels of edges outgoing a node share a common source state (due to hypothesis on labelling of paths), the labelling function lab can be consistently extended to nodes by letting $lab(n_0) = q_{in}$ and $lab(n) = q$ for any $(n, n') \in E$ with $lab(n', n) = (q', a, q)$.

Example 8.3. *On Figure 8.4, we have depicted a strategy pattern for the Broadcast Protocol of Figure 8.3.*

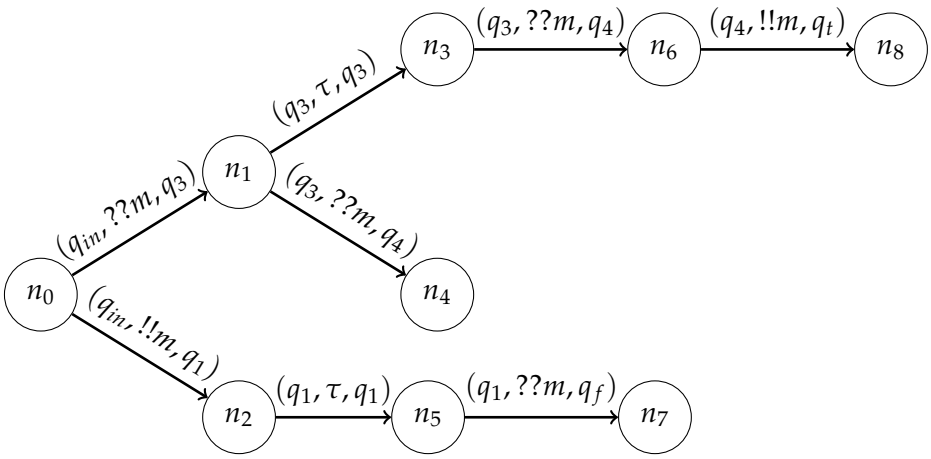

Figure 8.4: A strategy pattern for the Broadcast Protocol of Figure 8.3

Our aim is to reason on such strategy patterns to solve RBN-LOCAL-CONTROLREACH. First, remark that we can say that a local strategy σ for a Broadcast Protocol BP follows a strategy pattern T for the same protocol if for every path $e_1 \dots e_\ell$ in T (starting at n_0), we have that $lab(e_1) \dots lab(e_\ell)$ respects σ. By definition, any strategy pattern admits at least one local strategy. But we can as well refine the definition of strategy patterns such that they represent precisely local strategies yielding an execution which reaches a specific control state. We formalize this idea. Let $BP = (Q, M, \Delta, q_{in})$ be a Broadcast Protocol and $q_f \in Q$. A strategy pattern $T = (N, n_0, E, \Delta, lab)$ is said to be *q_f-admissible* if and only if there exists a strict total order $\prec \subseteq N \times N$ on the nodes of T such that:

1. $n \prec n'$ for all $(n, n') \in E$,
2. for all $e = (n, n') \in E$, if $lab(e) = (q, ??m, q')$ for some $q, q' \in Q$ and $m \in M$, then there exists $e_1 = (n_1, n_1') \in E$ such that $n_1' \prec n'$ and $lab(e_1) = (q_1, !!m, q_1')$ for some $q_1, q_1' \in Q$.
3. if $n \in N$ is such that for all $n' \in N$, $n \neq n'$ implies $n' \prec n$, then $lab(n) = q_f$.

In other words, the first condition states that $\prec$ respects the order of the tree T, the second condition that every node corresponding to a reception of a message m should be preceded by a node corresponding to the broadcast of this message and the last condition that the state q_f can be reached.

Example 8.4. *If we take the strategy pattern T represented on Figure 8.3 and we consider the following strict total order $n0 \prec n2 \prec n1 \prec n3 \prec n4 \prec n5 \prec n6 \prec n8 \prec n7$, then it allows us to deduce that T is q_f admissible.*

In [BFS15], we have shown that given a Broadcast Protocol $BP = (Q, M, \Delta, q_{in})$ and a control state $q_f \in Q$, there exist an execution $\rho := \lambda_{in} \Rightarrow^* \lambda$ in $RBN(BP)$ and a local strategy σ such that ρ respects σ and $\#q_f(\lambda) > 0$ if and only if there exists a q_f-admissible strategy pattern whose size is polynomial in the size of BP. This allows us to obtain an NP-algorithm for RBN-Local-ControlReach by guessing a strategy pattern together with the order characterising its q_f-admissibility. Similarly, by adapting the notion of admissibility we obtain an NP algorithm for RBN-Local-Target. A reduction from 3SAT allows us to get the matching lower bounds.

Theorem 8.2. *[BFS15]* RBN-Local-ControlReach *and* RBN-Local-Target *are* NP*-complete problems.*

Remark.

1. *The proof techniques developed in [BFS15] allow us furthermore to bound the number of processes necessary to have an execution respecting a local strategy and exhibiting a specific control strate [resp. target state] by a polynomial in the size of the considered protocol.*

2. *In [BFS15], we have as well studied the existence of local strategies for the parameterised control state and target state reachability problems in Ad Hoc Networks restricted to complete configurations. We have proved that these two problems are undecidable (even if, under the same assumption, the parameterised control state reachability without local strategy is decidable as recalled in Section 7.3.2.2). However, decidability can be regained for the control state reachability if one consider complete Broadcast Protocols where for every state q and every message m, there exists an outgoing edge from q able to receive the message m. But in that case, the problem is non primitive recursive.*

8.3 VERIFICATION OF PROBABILISTIC RECONFIGURABLE BROADCAST NETWORKS

As we have seen in the previous section, Reconfigurable Broadcast Networks enjoy good algorithmic properties for what concerns the verification of reachability properties. In [BFS14], we have investigated an extension of this model with probabilities. There are different options and motivations to insert probabilities in Reconfigurable Broadcast Networks. A first option to consider could be that the reconfiguration of the networks is probabilistic, in other words the set of receivers of a broadcast is chosen according to a probabilistic distribution among the possible sets of receivers. This option would allow to model in a certain sense the probability of failure of the message reception. It is not the direction we followed but it could be worth investigating it. Another option consists in equipping broadcast protocols with probabilities. This choice allows to model a probabilistic choice at the level of the protocol. This feature is inspired by randomised distributed algorithms where probabilities are used locally to break symmetries in the behavior of the network.

In [BFS14], we have hence extended the model of Reconfigurable Broadcast Networks by allowing probabilistic internal actions; a process can change its internal state according to a probabilistic distribution. As a consequence, the semantics of such networks is given by an infinite state system with probabilistic and non-deterministic choices (due to the possible interleaving and the choices of receiver set for each broadcast). As we have seen in Section 6.3, the verification of infinite state Markov Decision Processes can be a tedious task however

in the case of Probabilistic Reconfigurable Broadcast Networks, the peculiar shape of the systems allow to ease the reasoning and to propose nice algorithms. We have indeed studied in this latter context the probabilistic version of the parameterised control state reachability problem by seeking for the existence of a scheduler resolving the non-determinism while maximising or minimising the probability to reach a configuration exhibiting a specific state. In [BFS14], we have focused on the qualitative aspects of this problem comparing the probabilities with 0 and 1.

8.3.1 *Probabilistic Reconfigurable Broadcast Networks*

8.3.1.1 *Probabilistic Broadcast Protocols*

We begin by presenting the syntax and semantics behind what we call Probabilistic Reconfigurable Networks. In this model, as for Reconfigurable Broadcast Networks, we assume that each entity executes the same protocol represented by a finite state machine and that we call Probabilistic Broadcast Protocol. The main difference with Broadcast Protocol lies in the fact that some internal actions ares probabilistic in the sence that they are equipped with a probabilistic distribution defining the successor state in the protocol. This aspect allows to model randomised choices at the protocol level. The definition of probabilistic protocol is very close to the one of Markov Decision Processes given in Section 6.3 but the semantics will be different, however we will here as well use a semantics with the intervention of two players the non-deterministic player (Player 1) and the probabilistic player (Player P).

Definition 8.1 (Probabilistic Broadcast Protocol)**.** *A* Probabilistic Broadcast Protocol *BP is a tuple* $(Q, Q_1, Q_P, M, \Delta, \Delta_P, q_{in})$ *where:*

- Q *is a finite set of control states partitioned into* Q_1 *(states of Player 1) and* Q_P *(states of Player P),*
- M *is a finite alphabet of messages,*
- $\Delta \subseteq (Q_1 \times \{!!m, ??m \mid m \in M\} \times Q_1) \cup (Q_1 \times \{\tau\} \times Q)$ *correspond to the non-deterministic edges labelled by broadcast actions,*
- $\Delta_P : Q_P \mapsto \texttt{Dist}(Q_1)$ *is the set of probabilistic edges, and,*
- $q_{in} \in Q_1$ *is the initial control state.*

As for Broadcast Protocols, the label τ is used for an internal action of an entity, i.e. an action that is done independently, $!!m$ corresponds to the emission of the message m and $??m$ corresponds to the reception of the message m. In this model, we assume that only internal actions can lead to a probabilistic choice.

Example 8.5. *On Figure 8.5, we have depicted a simple example of Probabilistic Broadcast Protocols in which Player P has only a single state,* q_P *from which the probability to go to* q_1 *is one half as the probability to go to* q_2*.*

8.3.1.2 *Semantics in terms of Markov Decision Process*

We can now present the semantics associated to Probabilistic Broadcast Protocols in terms of Markov Definition Process. This definition extends the one of Reconfigurable Broadcast Networks presented previously by taking into account the difference between non-deterministic

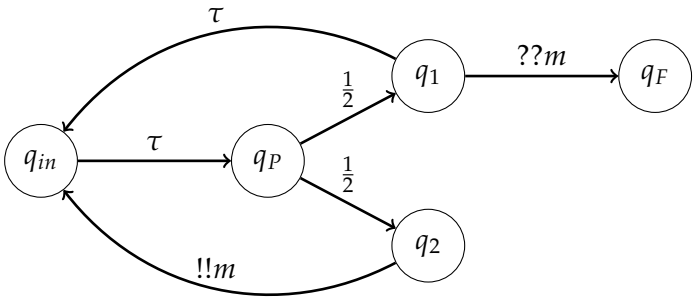

Figure 8.5: A Probabilistic Broadcast Protocol

and probabilistic choices. We suppose here that the choice of the active process (the one performing an internal action or a broadcast) is non-deterministic.

Let $BP = (Q, Q_1, Q_P, M, \Delta, \Delta_P, q_{in})$ be a Probabilistic Broadcast Protocol. The Player 1's configurations of the Probabilistic Reconfigurable Broadcast Network associated to BP are given by the set $\Upsilon_1 = \bigcup_{\ell \in \mathbb{N}\setminus\{0\}} \{v \in (Q_1 \times \{\bot, \top\})^\ell \mid card(\{p \in [1,\ell] \mid v[p] = (q, \top)\}) \leq 1\}$. Hence in these configurations, no process is labelled with a probabilistic state and at most one process has label $\top$, it corresponds to the process chosen to perform the next action, that we called the active process. The Player P's configurations are given by the set $\Upsilon_P = \bigcup_{\ell \in \mathbb{N}\setminus\{0\}} \{v \in (Q \times \{\bot\})^\ell \mid card(\{p \in [1,\ell] \mid v[p] = (q, \bot) \text{ and } q \in Q_P\}) = 1\}$. For this last set of configurations, the intuition is that when the network moves into a configuration of Υ_P from which it performs a probabilistic choice, this choice corresponds to the next possible state for the single process labelled by a state of Q_P (which is the one that has performed before an internal action leading to a state of Player P). We then denote by Υ the set of configurations $\Upsilon_1 \cup \Upsilon_P$. As for Broadcast Reconfiguration Networks, for a configuration $v \in \Upsilon$ such that $v \in (Q \cup \{\top, \bot\})^\ell$with $\ell \geq 1$, we denote by $\|v\| = \ell$ the number of entities/processes present in the network. The set of initial configurations is $\Upsilon_{in} = \bigcup_{\ell \in \mathbb{N}\setminus\{0\}} \{(q_{in}, \bot)\}^\ell$. Hence in an initial configuration, all processes are in state q_{in} and none of them is active.

The Probabilistic Reconfigurable Broadcast Network induced by BP is then given by the Markov Decision Process $PRBN(BP) = (\Upsilon, \Upsilon_1, \Upsilon_P, \Rightarrow, prob)$ where $\Rightarrow \subseteq \Upsilon_1 \times \Upsilon$ and $prob : \Upsilon_P \to \mathtt{Dist}(\Upsilon_1)$ are defined as follows. We have $v \Rightarrow v'$ if and only if $\|v\| = \|v'\|$ and one of the following conditions holds:

- **Process choice:** $card(\{p \in [1, \|v\|] \mid v[p] = (q, \top)\}) = 0$ and there exists $p \in [1, \|v\|]$ such that $v[p] = (q, \bot)$ and $v'[p] = (q, \top)$ and there exists $(q, a, q') \in \Delta$ with $a \in \{!!m \mid m \in M\} \cup \{\tau\}$ and $v[p'] = v'[p']$ for all $p' \in [1, \|v\|] \setminus \{p\}$;
- **Internal action:** there exists $p \in [1, \|v\|]$ such that $v[p] = (q, \top)$ and $v'[p] = (q', \bot)$ and $(q, \tau, q') \in \Delta$ and $v[p'] = v'[p']$ for all $p' \in [1, \|v\|] \setminus \{p\}$;
- **Broadcast communication:** there exist $p \in [1, \|v\|]$ and $m \in M$ such that $v[p] = (q, \top)$ and $v'[p] = (q', \bot)$ and $(q, !!m, q') \in \Delta$ and for all $p' \in [1, \|v\|] \setminus \{p\}$ either $v[p'] = (q'', \bot)$ and $v'[p'] = (q''', \bot)$ and $(q'', ??m, q''') \in \Delta$, or $v[p'] = (q'', \bot) = v'[p']$.

And for all $v \in \Upsilon_P$, if p is the unique process such that $v[p] = (q, \bot)$ with $q \in Q_P$ then for all $v' \in \Upsilon_1$ we have that if $\|v\| = \|v'\|$ and $v'[p] = (q', \bot)$ and $v[p'] = v'[p']$ for all $p' \in [1, \|v\|] \setminus \{p\}$ then $prob(v)(v') = \Delta_P(q)(q')$ and otherwise $prob(v)(v') = 0$.

Example 8.6. *On Figure 8.6, we have depicted some finite plays of the Markov Decision Process $PRBN(BP)$ induced by the Probabilistic Broadcast Protocol BP of Figure 8.5. All these plays start from the initial configuration with three processes and we have represented by dotted edge the probabilistic transitions.*

8.3.1.3 *Qualitative reachability problems*

We now present the verification problems we have studied in [BFS14] over Probabilistic Reconfigurable Broadcast Networks. it corresponds to the probabilistic version of the parameterised control state reachability we have studied previously in Ad Hoc Networks and Reconfigurable Broadcast Networks. We have only obtained results for qualitative questions (where we compare the probabilities with 0 and 1) and computing the exact probabilities seem to be a tedious task. Before to define the problem, we need an extra notation. Given a Probabilistic Broadcast Protocol $BP = (Q, Q_1, Q_P, M, \Delta, \Delta_P, q_{in})$ and a control state $q_F \in Q$, we denote by $[\![\mathrm{F} q_F]\!]$ the set of maximal plays of $PRBN(BP)$ given by $\{v_0 v_1 v_2 \dots \mid \exists i.\exists j.j \leq \|v_i\|$ and $v_i[j] = (q_F, \bot)\}$, i.e. the set of maximal plays reaching a configuration where one of the process is labelled by $(q_F, \bot)$. The different qualitative reachability problems we analysed are parameterised by an option $\mathtt{opt} \in \{inf, sup\}$ specifying if we wish to minimise or maximise the probability of reaching the state, by a constant $\mathtt{b} \in \{0, 1\}$ and by $\sim \in \{<, =, >\}$ and they are given by the following common definition:

PRBN-ControlReach$^{\sim \mathtt{b}}_{\mathtt{opt}}$

Input:	A Probabilistic Broadcast Protocol $BP = (Q, Q_1, Q_P, M, \Delta, \Delta_P, q_{in})$ and a control state $q_F \in Q$;
Question:	Does there exists an initial configuration $v_{in} \in \mathrm{Y}_{in}$ such that $\mathbb{P}^{\mathtt{opt}}(PRBN(BP), v_{in}, [\![\mathrm{F} q_F]\!]) \sim \mathtt{b}$?

Note that as the previous problems presented in the framework of parameterised networks, here again we are seeking for an initial configuration in an infinite set. However, if we fix the initial configuration, then the problem boils down to the analysis of a finite state Markov Decision Process.

8.3.2 *Reconfigurable Broadcast Networks Games*

In order to solve some of the parameterised qualitative problems mentioned before, as we have done in the case of VASS-MDP (see Section 6.3), we will here as well propose a reduction towards a two player game. This technique is not classical and in [Cha+09], the authors show the connection between qualitative questions in Markov Decision Processess and fixpoint logics (for which the model-checking can be solved thanks to games). However, what makes the task hard in our context is that we need to define a game on parameterised networks and then to propose a method to solve it, knowing that we deal with an infinite system. We will show in this section how we have faced these two issues in [BFS14].

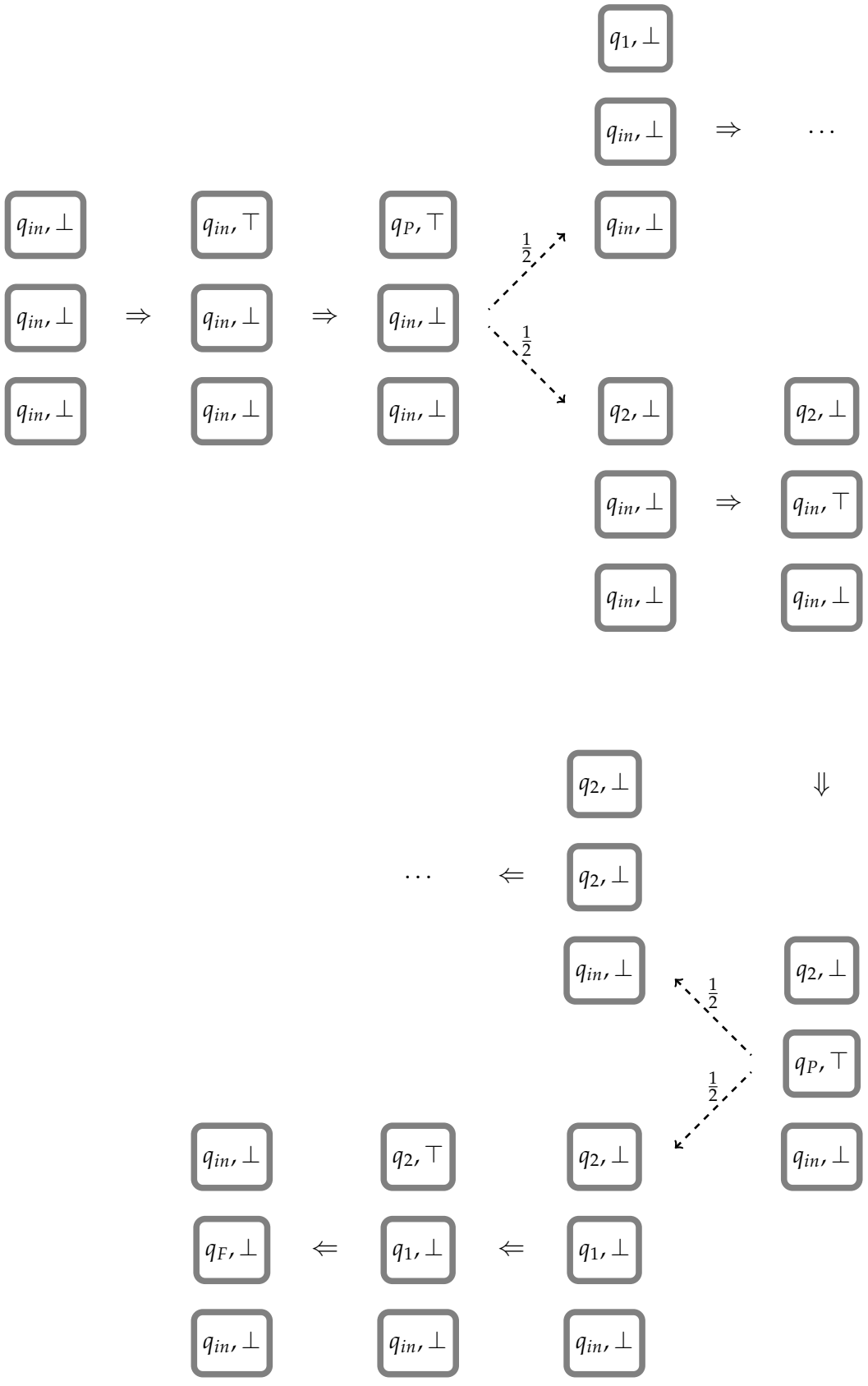

Figure 8.6: Example of plays in $PRBN(BP)$ for the Probabilistic Broadcast Protocol BP of Figure 8.5

8.3.2.1 *Playing games on Reconfigurable Broadcast Networks*

We first need to propose a new definition of a Broadcast Protocol where the set of states is divided among two players and to which we add some colouring functions to deal with parity conditions. We will as well equip our protocol with a subset of edges, we will call safe edges, which correspond to edges that Player 1 can take and if an edge not belonging to this set is taken then Player 1 will lose. This extra condition will be useful to solve for instances PRBN-ControlReach$^{=0}_{\texttt{min}}$ where we want to find a scheduler that can avoid to go to the target state.

Definition 8.2 (Parity Broadcast Protocol)**.** *A* Parity Broadcast Protocol *BP is a tuple* $(Q, Q_1, Q_2, M, \Delta, q_{in}, col, safe)$ *where:*

- Q *is a finite set of control states partitioned into* Q_1 *(states of Player 1) and* Q_2 *(states of Player 2),*
- M *is a finite alphabet of messages,*
- $\Delta \subseteq (Q_1 \times (\{!!m, ??m \mid m \in M\} \cup \{\tau\}) \times Q) \cup (Q_2 \times \{\tau\} \times Q)$ *correspond to the edges labelled by broadcast actions,*
- $q_{in} \in Q_1$ *is the initial control state,*
- $col : \Delta \mapsto \mathbb{N}$ *is the colouring function, and,*
- $safe \subseteq \Delta$ *is the set of safe edges.*

We remark that we associate in our context colours to edges and not to states because since we deal with networks where each process is in a state of the Parity Broadcast Protocol, having colours associated to states would mean to deal with set of colours which might be difficult. On the other hand, having colours associated to edges allows us to focus on the sequence of colours generated by the active transitions (internal ones or broadcasts of a message) in an execution. Note as well that the roles of Player 1 and Player 2 are not symmetric: only Player 1 can initiate a communication and Player 2 performs only internal actions.

The semantics associated to a Parity Broadcast Protocol is given in term of a 2 Player game whose definition is similar to the Markov Devision Process induced by a Probabilistic Broadcast Protocol. Let $BP = (Q, Q_1, Q_2, M, \Delta, q_{in}, col, safe)$ be a Parity Broadcast Protocol. Here as well, the Player 1 has the ability to chose a process which will perform an action and according to the control state labelling the process, either Player 1 or Player 2 will then perform the next move. The Player 1's configurations of the Reconfigurable Broadcast Network Game associated to BP are given by the set $Y_1 = \bigcup_{\ell \in \mathbb{N}\setminus\{0\}} (Q \times \{\bot\})^\ell \cup \{v \in (Q \times \{\bot, \top\})^\ell \mid card(\{p \in [1, \ell] \mid v[p] = (q, \top) \text{ and } q \in Q_1\}) = 1 \text{ and } card(\{p \in [1, \ell] \mid v[p] = (q, \top) \text{ and } q \in Q_2\}) = 0\}$ and the Player 2's configurations are given by the set $Y_2 = \bigcup_{\ell \in \mathbb{N}\setminus\{0\}} \{v \in (Q \times \{\bot, \top\})^\ell \mid card(\{p \in [1, \ell] \mid v[q] = (q, \top) \text{ and } q \in Q_2\}) = 1 \text{ and } card(\{p \in [1, \ell] \mid v[q] = (q, \top) \text{ and } q \in Q_1\}) = 0\}$ and we set $Y = Y_1 \cup Y_2$. For $v \in Y$ such that $v \in (Q \cup \{\top, \bot\})^\ell$with $\ell \geq 1$, we use again the notation $\|v\|$ to denote ℓ the number of entities/processes present in the network and the set of initial configurations is $Y_{in} = \bigcup_{\ell \in \mathbb{N}\setminus\{0\}} \{(q_{in}, \bot)\}^\ell$.

The Reconfigurable Broadcast Networks Game induced by BP is then given by the two player game $RBNG(BP) = (Y, Y_1, Y_2, \Rightarrow, col, safe)$ where $\Rightarrow \subseteq Y \times Y$, $Col :\Rightarrow \mapsto \mathbb{N}$ and $Safe \subseteq \Rightarrow$ are defined as follows. We have $v \Rightarrow v'$ if, and only if, $\|v\| = \|v'\|$ and one the following conditions holds:

- **Process choice:** $card(\{p \in [1, \|v\|] \mid v[p] = (q, \top)\}) = 0$ and there exists $p \in [1, \|v\|]$ such that $v[p] = (q, \bot)$ and $v'[p] = (q, \top)$ and there exists $(q, a, q') \in \Delta$ with $a \in \{!!m \mid m \in M\} \cup \{\tau\}$ and $v[p'] = v'[p']$ for all $p' \in [1, \|v\|] \setminus \{p\}$; in that case $col(v, v') = 0$ and $(v, v') \in Safe$;

- **Internal action:** there exists $p \in [1, \|v\|]$ such that $v[p] = (q, \top)$ and $v'[p] = (q', \bot)$ and $(q, \tau, q') \in \Delta$ and $v[p'] = v'[p']$ for all $p' \in [1, \|v\|] \setminus \{p\}$; in that case $Col(v, v') = col((q, \tau, q'))$ and $(v, v') \in Safe$ if and only if $(q, \tau, q') \in safe$;

- **Broadcast communication:** there exist $p \in [1, \|v\|]$ and $m \in M$ such that $v[p] = (q, \top)$ and $v'[p] = (q', \bot)$ and $(q, !!m, q') \in \Delta$ and for all $p' \in [1, \|v\|] \setminus \{p\}$ either $v[p'] = (q'', \bot)$ and $v'[p'] = (q''', \bot)$ and $(q'', ??m, q''') \in \Delta$, or $v[p'] = (q'', \bot) = v'[p']$; in that case $Col(v, v') = col((q, !!m, q'))$ and $(v, v') \in Safe$ if and only if $(q, !!m, , q') \in safe$ and for all $(q'', ??m, q''')$ used we have $(q'', ??m, q''') \in safe$.

Note hence that in our definition only the colours of the edges labelled by a broadcast or an internal action matter.

In order to define how the game is played between Player 1 and Player 2, we need further notations. A finite play ρ in $RBNG(BP)$ is a finite sequence of configurations $v_0 v_1 \dots v_n \in \Upsilon^*$ such that $v_i \Rightarrow v_{i+1}$ for all $0 \leq i < n$ and $v_0 \in \Upsilon_{in}$; it is said to be maximal if there does not exists $v \in \Upsilon$ such that $v_n \Rightarrow v$, in other words v_n is a deadlock configuration. An infinite play is an infinite sequence $\rho \in \Upsilon^\omega$ such that any prefix of ρ is a finite play. Maximal plays of $RBNG(BP)$ are either finite maximal plays or infinite plays.

A strategy for Player id, with $id \in \{1, 2\}$, is a function $\pi : \Upsilon^* \cdot \Upsilon_{id} \mapsto \Upsilon$ such that for every finite play $\rho = v_0 v_1 \dots v_n$ with $v_n \in \Upsilon_{id}$, we have $v_n \Rightarrow \pi(\rho)$. Given an initial configuration v_{in}, a strategy π_1 for Player 1 and a strategy π_2 for Player 2, there exists an unique maximal play $\texttt{play}(v_{in}, \pi_1, \pi_2) = v_0 v_1 \dots$ such that $v_0 = v_{in}$ and for all $i \in \mathbb{N}$ if v_{i+1} exists and $v_i \in \Upsilon_{id}$ for $id \in \{1, 2\}$ then $\pi_{id}(v_0 v_1 \dots v_i) = v_{i+1}$.

We should now define the winning objectives for Player 1 we consider. A winning objective is a subset of maximal plays $W \subseteq \Upsilon^* \cup \Upsilon^\omega$ and we will look at two different winning objectives:

1. The parity objective: $W_P = \{v_0 v_1 \dots \text{ is an infinite play} \mid \max(n \in \mathbb{N} \mid \forall i \geq 0. \exists j \geq i. Col((v_j, v_{j+1})) = n) \text{ is even } \}$

2. The safety parity objective: $W_{SP} = \{v_0 v_1 \dots \in W_P \mid \forall i \geq 0. (v_i, v_{i+1}) \in Safe\} \cup \{\rho = v_0 \dots v_n \mid \rho \text{ is a finite play and } \forall 0 \leq i < n. (v_i, v_{i+1}) \in Safe\}$

Hence the parity objective denotes the infinite plays for which the maximal color visited infinitely often is even and the safety parity objectives represents the infinite plays satisfying the parity objective which only use edges in $safe$ and the finite plays which only use edges in $safe$. Finally, we say that v_{in} is a winning configuration for Player 1 for an objective W if and only if there exists a strategy π_1 for Player 1 such that for all strategies π_2 of Player 2, $\texttt{play}(v_{in}, \pi_1, \pi_2) \in W$. In that case, $\texttt{play}(v_{in}, \pi_1, \pi_2)$ is called a winning play for W.

Thanks to these previous definitions, we can define the parameterised parity game problem for Reconfigurable Broadcast Networks Games as follows:

PARITY-RBN-GAME	
Input:	A Parity Broadcast Protocol BP;
Question:	Does there exists an initial configuration $v_{in} \in Y_{in}$ such that v_{in} is a winning configuration for Player 1 for W_P?

Similarly, if instead of W_P, we consider the safety parity objective, this gives rise to the following problem.

SAFEPARITY-RBN-GAME	
Input:	A Parity Broadcast Protocol BP;
Question:	Does there exists an initial configuration $v_{in} \in Y_{in}$ such that v_{in} is a winning configuration for Player 1 for W_{SP}?

8.3.2.2 *Solving Reconfigurable Broadcast Network Games*

In order to solve Reconfigurable Broadcast Network Games for both parity and safety parity objectives, we first show in [BFS14] that we can restrict the strategies of Player 2 to strategies that always choose from a given control state the same successor independently of the configuration or the history in the game.

It is well-known for the case of parity games over a finite state arena, that it is enough to consider memoryless strategies for both Player 1 and Player 2 (memoryless in the sense that in each state of the arena, when a play reaches this state, the owner of the state always choose the same successor state) to find whether a configuration is winning or not, see for instance [Zie98; GTW02]. Having such a result allows to design a simple algorithm for verifying if a given state of the arena is winning for Player 1 which consists in guessing a memoryless strategy for Player 1 (there is a finite number of such strategies) and then check in the game arena whether there exists a play following this strategy that does not respect the parity condition (this latter condition being easier to check because we only have to deal with the choices of Player 2).

In the context of Reconfigurable Broadcast Network Games, since when the number of entities in the network is given, the number of possible configurations is finite, we could derive a similar property, stating that it is enough to consider memoryless strategies, which always choose the same successor configuration from a given configuration. However the difficulty here lies one again in that we do not know the size of the initial configuration and we do not have a way to represent memoryless strategies which should take into account an infinite set of configurations. This is the reason why finding more restrictions on the allowed strategies is necessary for these games.

We move now to the definition of this specific strategies. We consider a Parity Broadcast Protocol $BP = (Q, Q_1, Q_2, M, \Delta, q_{in}, col, safe)$. A *local behavior* for Player 2 is a function $b : Q_2 \mapsto Q$ such that $(q, \tau, b(q)) \in \Delta$ for all $q \in Q_2$. This local behavior indicates to Player 2 from each of its state which successor state it should choose. Such a local behavior induces a *state strategy* π_b for Player 2 defined as follows. For every finite play $\rho = v_0 v_1 \dots v_n$ with $v_n \in Y_2$, we have $\pi_b(\rho) = v'$ if and only if $v_n[p] = (q, \top)$ for some (unique) $p \in [1, \|v_n\|]$ and $v_n[p] = (b(q), \bot)$ and $v_n \Rightarrow v$, in other words, if p is the unique process which should

perform the action and whose state belongs to Q_2 then this process will change its state accordingly to the local behavior b. Note that the number of state strategies for BP is finite and each strategy can be represented by a local behavior which has a polynomial size in the size of BP.

The main proposition which allows then to solve Reconfigurable Broadcast Network Games can then be stated as follows.

Proposition 8.1. *[BFS14] For $W \in \{W_P, W_{SP}\}$, there exists an initial configuration $v_{in} \in Y_{in}$ such that v_{in} is a winning configuration for Player 1 for W if and only if for all state strategies π_2 of Player 2, there exists an initial configuration $v_{in} \in Y_{in}$ and a strategy π_1 for Player 1 such that $\texttt{play}(v_{in}, \pi_1, \pi_2) \in W$.*

This proposition shows us first that we can restrict our attention to state strategies for Player 2 to solve the Reconfigurable Broadcast Network Games, but as well it provides a methodology to solve the game. Indeed, the last task to show is that in a game where Player 2 has no choice, we can decide the existence of maximal plays satisfying the winning condition.

To show that there exists a finite maximal plays which only uses edges in $safe$, it suffices to show that there exists an initial configuration from which one can reach a configuration where none of the processes is in a state from which it can perform a broadcast or an internal action (without taking edges that are not safe). This last property can be verified in NP for Reconfigurable Broadcast Networks as we have shown with Theorem 8.1.

To show that there exists an infinite plays which belongs W_{SP}, we follow the following steps which can be achieved in polynomial time:

1. We remove the unsafe edges from the protocol;
2. We compute in polynomial time the reachable control states, as explained in Section 8.1.3.1 and in [Del+12] and we know from [Del+12] that there exists a reachable configuration exhibiting as many as this reachable control states as we want;
3. We look for an infinite loop respecting the parity condition from such a configuration which can be done in polynomial time (it boils down to find a positive cycle in a VASS [KS88]).

Consequently, we deduce that to find the existence of a maximal play satisfying W_P can be done in polynomial time and if one looks for a maximal play satisfying W_{SP} it can be achieved by a NP algorithm. To solve Reconfigurable Broadcast Networks Games, the strategy consists in guessing a state strategy for Player 2, and then verifying that there does not exist a maximal plays satisfying the winning objective. Using Proposition 8.1, we obtain the following result.

Theorem 8.3. *[BFS14]* Parity-RBN-Game *is in* co-NP *and* SafeParity-RBN-Game *is in* $\Pi_2^p (= \text{co-NP}^{\text{NP}})$.

8.3.3 *Solving the qualitative reachability problems*

In [BFS14], we provide the precise complexity of all the relevant qualitative reachability problems for Probabilistic Reconfigurable Broadcast Networks. For some cases, the answer

was easy to provide, for other we relies on a reduction towards Reconfigurable Broadcast Network Games and used the result of Theorem 8.3.

We first present the easy cases. First for PRBN-ControlReach$^{>0}_{sup}$, this problem is intereducible to the parameterised control state reachability problem in Reconfigurable Broadcast Networks which we know to be P-complete (see Theorem 8.1). For PRBN-ControlReach$^{=0}_{sup}$, PRBN-ControlReach$^{<1}_{sup}$, PRBN-ControlReach$^{=1}_{inf}$ and PRBN-ControlReach$^{>0}_{inf}$ we use a monotonicity property: intuitively with more processes the probability to reach the target state can only increase, hence these problems reduce to the qualitative reachability problems in the finite state Markov Decision Process representing the behavior of the network with a single process and which can be solved in polynomial time (see e.g. [BK08]).

Theorem 8.4. *[BFS14]* PRBN-ControlReach$^{>0}_{sup}$, PRBN-ControlReach$^{=0}_{sup}$, PRBN-ControlReach$^{<1}_{sup}$, PRBN-ControlReach$^{=1}_{inf}$ *and* PRBN-ControlReach$^{>0}_{inf}$ *are in* Ptime.

We shall now explain how we solve the other cases thanks to the results on Reconfigurable Broadcast Network Games. We present the most involved case which is PRBN-ControlReach$^{=1}_{sup}$. From a Probabilistic Broadcast Protocol $BP = (Q, Q_1, Q_P, M, \Delta, \Delta_P, q_{in})$ and a control state $q_F \in Q$, we derive the Parity Broadcast Protocol $BP' = (Q', Q'_1, Q'_2, M, \Delta', q'_{in}, col, safe)$ such that:

- $Q' = Q'_1 \cup Q'_2$ and $Q'_1 = Q_1 \cup (Q_P \times \{1\})$ and $Q'_2 = Q_P \times \{2\}$,
- Δ' is the smallest set containing the following sets of edges:
  1. $Q_1 \times \{!!m, ??m \mid m \in M\} \cup \{\tau\} \times Q_1) \cap \Delta$,
  2. $\{(q_F, \tau, q_F)\}$,
  3. $\{(q, \tau, (q', 2)), ((q', 2), \tau, (q', 1)) \mid q' \in Q_P, (q, \tau, q') \in \Delta, i \in \{1, 2\}\}$,
  4. $\{(q, i), \tau, q') \mid q \in Q_P, \Delta_P(q)(q') > 0, i \in \{1, 2\}\}$,
- $col((q_F, \tau, q_F)) = 2$ and $col((q, 2), \tau, q') = 2$ and $col(\delta) = 1$ for all other edges, and,
- $safe = \Delta'$.

Here is the intuition behind this construction. All random choices are replaced in BP' with choices for Player 2, where either he decides the outcome of the probabilistic choice taking a transition of the form $((q, 2), \tau, q')$ or he lets Player 1 perform the choice taking a transition of the form $((q, 2), \tau, (q, 1))$. And only transitions where Player 2 makes the decision corresponding to a probabilistic choice and the self loop on the state q_F have colours 2 whereas all the other edges have colour 1. Hence if there is an initial configuration v_{in} in the Probabilistic Reconfigurable Broadcast Network such that $\mathbb{P}^{\sup}(PRBN(BP), v_{in}, [\![\mathbf{F} q_F]\!]) = 1$, then v_{in} is a winning configuration for Player 1 for the parity objective in the Reconfigurable Broadcast Network Game associated to BP'. The intuition being that to win in this game, from each reachable configuration Player 1 should be able to reach a configuration exhibiting the control state q_F (indeed the only way for Player 1 to lose would be that at some point Player 2 always let him do the probabilistic choice and that he cannot reach q_F from this moment on), but this last sentence is exactly the characterisation of having $\mathbb{P}^{\sup}(PRBN(BP), v_{in}, [\![\mathbf{F} q_F]\!]) = 1$ in the finite state Markov Decision Process corresponding to the network with v_{in} as started configuration. We show as well that is if there exists an

initial winning configuration v'_{in} for the parity objective in $RBNG(BP')$, then there exists an initial configuration v_{in} such that $\mathbb{P}^{\sup}(PRBN(BP), v_{in}, [\![\mathrm{F} q_F]\!]) = 1$, however this direction is more complex. This reasonning tells us that there is a logarithmic space solution from PRBN-ControlReach$^{=1}_{sup}$ to Parity-RBN-Game. We show as well in [BFS14] that PRBN-ControlReach$^{=1}_{sup}$ is co-NP-hard by reducing the unsatisifiability problem for propositional logic formulas.

Example 8.7. *On Figure 8.7, we have depicted the Parity Broadcast Protocol BP' associated to the Probabilistic Broadcast Protocol BP given on Figure 8.5 to solve* PRBN-ControlReach$^{=1}_{sup}$. *On the example, we have provided the colour associated to each edge after the action. On this example, we can see that the initial configuration v_{in} with two processes both in $(q_{in}, \bot)$ is winning for Player 1 for the parity objective W_P. The strategy of Player 1 consists in first choosing always the first process as the active one, until it is put in the state q_1 (this will necessarily happens either because Player 2, will put him in q_1 or at some point Player 2 will put this process in state $(q_P, 1)$ from which Player 1 will choose to move it to q_1). Then Player 1 can force the second process to be eventually in state q_2 and finally the second process can broadcast the message m received by the first process which geos to q_F. Consequently, we have as well that $\mathbb{P}^{\sup}(PRBN(BP), v_{in}, [\![\mathrm{F} q_F]\!]) = 1$.*

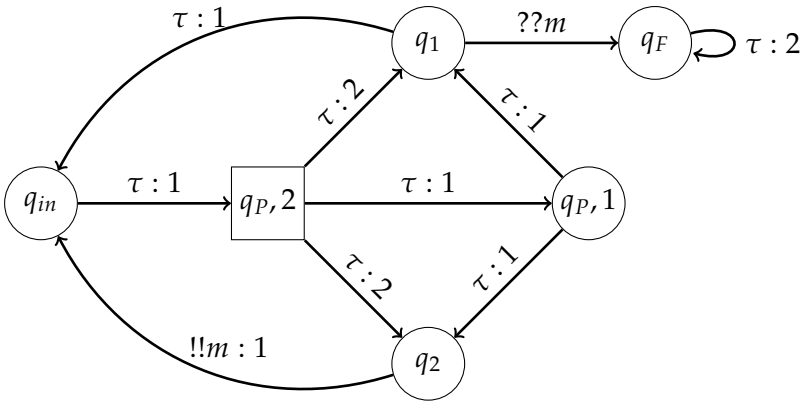

Figure 8.7: The Parity Broadcast Protocol built from the Probabilistic Broadcast Protocol of Figure 8.5 to solve PRBN-ControlReach$^{=1}_{sup}$

For the decision problems PRBN-ControlReach$^{=0}_{inf}$ and PRBN-ControlReach$^{<1}_{inf}$, we can both reduce them similarly to SafeParity-RBN-Game, the reduction being more intuitive. For PRBN-ControlReach$^{=0}_{inf}$, from a Probabilistic Broadcast Protocol BP, we build a Parity Broadcast Protocol where all random choices are replaced by choices for Player 2 and the transitions leading to the target state q_F are the only one not to be safe,because Player 1 wants to avoid this state (in this case all the edges have the same colour 0). For PRBN-ControlReach$^{<1}_{inf}$, we build a Parity Broadcast Protocol consisting of two copies of the Probabilistic Broadcast Protocol. In the fist copy, all random choices are replaced with choices of Player 1, whereas in the second copy they are replaced with choices of Player 2 and at any time, it is possible to move from the first copy to the second copy. In this construction, the colours of edges with target in the second copy is 2 and all the other edges have colour 1 and the only unsafe edges are those leading to q_F.

All these reductions, together with the result of Proposition 8.3 lead to the following theorem.

Theorem 8.5. *[BFS14]*

- PRBN-CONTROLREACH$^{=1}_{sup}$ *is* CO-NP*-complete.*
- PRBN-CONTROLREACH$^{=0}_{inf}$ *and* PRBN-CONTROLREACH$^{<1}_{inf}$ *are in* $\Pi^p_2 (= \text{CO-NP}^{\text{NP}})$.

Remark. *Note that as one can see in [Fou15], if there is no reachable deadlock configuration in the network associated to Parity Broadcast Protocol or to Probabilistic Broadcast Protocol, then the complexity of* SAFEPARITY-RBN-GAME *and of* PRBN-CONTROLREACH$^{=0}_{inf}$ *and* PRBN-CONTROLREACH$^{<1}_{inf}$ *becomes* CO-NP*. Indeed when looking at the resolution method for* SAFEPARITY-RBN-GAME*, the* NP *upper bound comes from the search of the finite maximal plays. In the general case, to the best of my knowledge, it is not known whether these two problems are complete for the class* Π^p_2 *or if better upper bounds can be found.*

8.4 VERIFICATION OF RECONFIGURABLE BROADCAST NETWORKS WITH REGISTERS

In all the variants of Reconfigurable Broadcast Networks we have presented so far, the different entities in the networks communicate via broadcasted messages which belong to a finite alphabet. In [DST13; DST16], we have extended the original model in order to take into account messages belonging to an infinite alphabet and we have furthermore assumed that each entity in the network is equipped with a finite set of registers allowing to store the exchanged values. This research is strongly inspired by the model of register automata introduced in [KF94]. This latter model extends finite state automata with a storing mechanism, also called registers, used to store a data and compare it with future values and it recognises words over a data alphabet where at each position there is a pair composed of a letter from a finite alphabet and a letter from an infinite alphabet called the data. In our context, we use as well registers to store data and to compare it with future values, the main difference being that the values do not appear in a word being read but are exchanged between the entities. As for Reconfigurable Broadcast Networks, we then study the parameterised control state reachability problem for this model asking whether there exists an initial configuration from which it is possible to reach a configuration exhibiting a specific control state.

8.4.1 *Reconfigurable Broadcast Networks with Registers*

8.4.1.1 *Syntax*

As in the previous models, each entity in our networks executes the same finite state protocol whose edges are labelled with operations over a finite set of registers. An entity (we may call equivalently node or process) can then transmit part of its current registers to other entities thanks to broadcast. For this matter, we assume that the exchanged messages carry both a type and a finite tuple of data and upon reception, a receiver can test/store/ignore the data contained inside the message.

We first present the set of 'actions' which label the edges of the protocols. We use $R \geq 0$ to denote the number of registers in each node and $F \geq 0$ to denote the number of data fields available in each message. Furthermore, the message type will be given by a finite

message alphabet M. The set of Broadcast Actions with registers parameterised by R, F and M is then given by:

$$\texttt{Bcast}_{R,F}(M) = \{!!m[r_1, \ldots, r_F] \mid m \in M \text{ and } r_i \in [1, R] \text{ for all } i \in [1, F]\}$$

The action $!!m[r_1, \ldots, r_F]$ corresponds to the broadcast of a message of type m whose i-th field (for $i \in [1, F]$) contains the value of the r_i-th register of the emitter. For instance for $R = 2$ and $F = 4$, the action $!!\texttt{req}(1, 1, 2, 1)$ sends a message of type `req` in which the current value of the first register of the sender is copied in the two first fields and in the last field of the message and the value of the second register of the sender is copied into the third field of the message.

On reception of a message, an entity can then either compare the value of a message field against the current value of a register or store the value of a message field in a register or ignore a message field. The set of Receive Actions with registers parameterised by R, F and M is then given by:

$$\texttt{Rec}_{R,F}(M) = \left\{ ??m[\alpha_1, \ldots, \alpha_F] \;\middle|\; \begin{array}{l} m \in M \text{ and } \alpha_i \in \texttt{RegAct}_R \text{ for all } i \in [1, F] \\ \text{and if } \alpha_i = \alpha_j = \downarrow r \text{ then } i = j \end{array} \right\}$$

where $\texttt{RegAct}_r = \{?r, ?\bar{r}, \downarrow r, * \mid r \in [1, R]\}$ is the set of Register Actions. When used in a given position of a receive action, $?r$ [resp. $?\bar{r}$] tests whether the content of the r-th register of the receiver is equal [resp. different] to the corresponding value stored in the corresponding field of the message, $\downarrow r$ is used to store the value present in the field of the message in the r-th register and $*$ is used to ignore the value present in the field of the message. For instance for $R = 2$ and $F = 4$, the action $!!\texttt{req}(?\bar{2}, ?1, *, \downarrow 1)$ is used to receive a message of type `req` in which the value of the first field is tested for inequality against the value present in the second register of the receiver, the value of the second field is tested for equality against the value present in the first register of the receiver, the third field is ignored and the value stored in the fourth field is stored in the first register.

We have now all the elements to define Register Broadcast Protocols which extend Broadcast Protocols and which will be executed by nodes in the network equipped with registers and where the transmitted messages will have some fields to store values.

Definition 8.3 (Register Broadcast Protocol)**.** *A (R, F)-Register Broadcast Protocol BP is a tuple (Q, M, Δ, q_{in}) where:*

- *Q is a finite set of control states,*
- *M is a finite alphabet of messages,*
- *$\Delta \subseteq Q \times (\texttt{Bcast}_{R,F}(M) \cup \texttt{Rec}_{R,F}(M) \cup \{\tau\}) \times Q$ is a finite set of edges labelled by broadcast and receive actions, and,*
- *$q_{in} \in Q$ is the initial control state.*

8.4.1.2 *Semantics*

To a (R, F)-Register Broadcast Protocol $BP = (Q, M, \Delta, q_{in})$, we associate a Data Reconfigurable Broadcast Network whose configurations are pairs $(\lambda, \mathbf{M})$ in $\bigcup_{\ell \in \mathbb{N} \setminus \{0\}} Q^\ell \times (\mathbb{N}^R)^\ell$. As for Reconfigurable Broadcast Networks, for a configuration $\zeta = (\lambda, \mathbf{M}) \in Q^\ell \times (\mathbb{N}^R)^\ell$ with $\ell \geq 1$, we denote by $\|\zeta\| = \ell$ its dimension, corresponding to the number of entities/processes present in the network and, for each $1 \leq p \leq \|\zeta\|$, $\lambda[p]$ represents the state

of the p-th process wheras $\mathbf{M}[p] \in \mathbb{N}^R$ represents the memory (the contents of the registers) of the p-th process. A configuration $\zeta = (\lambda, \mathbf{M})$ is said to be initial if and only if $\lambda[p] = q_{in}$ for all $p \in [1, \|\zeta\|]$ and for all $p, p' \in [1, \|\zeta\|]$ and all $r, r' \in [1, R]$, if $p \neq p'$ or $r \neq r'$, we have $\mathbf{M}[p][r] \neq \mathbf{M}[p'][r']$. In other words, in an initial configuration, all processes are in the initial state of the protocol and no register share the same value. We denote by Z the set of configurations and by Z_{in} the set of initial configurations.

The Data Reconfigurable Broadcast Network induced by the (R, F)-Register Broadcast Protocol $BP = (Q, M, \Delta, q_{in})$ is given by The transition system $DRBN(BP) = (Z, \Rightarrow)$ where $\Rightarrow \subseteq Z \times Z$ is the transition relation which we shall now define. We have $\zeta \Rightarrow \zeta'$ if and only if $\|\zeta\| = \|\zeta'\|$ and there exists $p \in [1, \|\zeta\|]$ such that if $\zeta = (\lambda, M)$ and $\zeta' = (\lambda', M')$ one the following conditions holds:

- **Internal action:** there exists $(\lambda[p], \tau, \lambda'[p]) \in \Delta$ and $M[p] = M'[p]$ and for all $p' \in [1, \|\zeta\|] \setminus \{p\}$, we have $\lambda[p'] = \lambda'[p']$ and $M[p'] = M'[p']$;
- **Broadcast communication:** there exists $\delta = (\lambda[p], !!m[r_1, \ldots, r_F], \lambda'[p]) \in \Delta$ and $M[p] = M'[p]$ and for all $p' \in [1, \|\zeta\|] \setminus \{p\}$, we ave either ($\lambda[p'] = \lambda'[p']$ and $M[p'] = M'[p']$), or, there exists $\delta = (\lambda[p'], ??m[\alpha_1, \ldots, \alpha_F], \lambda'[p']) \in \Delta$ such that
  1. for all $i \in [1, F]$, if there exists $r \in [1, R]$ such that $\alpha_i = ?r$ [resp.$\alpha_i = ?\bar{r}$] then $M[p'][r] = M[p][r_i]$ [resp. $M[p'][r] \neq M[p][r_i]$], and,
  2. for all $r \in [1, R]$, if there exists $i \in [1, F]$ such that $\alpha_i = \downarrow r$ then $M'[p'][r] = M[p][r_i]$ and othewise $M'[p'][r] = M[p'][r]$.

Hence if a broadcast communication is performed, a node can receive it if the values of its register pass the tests (given by $?r$ and $?\bar{[}r]$ in the description of the reception) and afterwards he can store received values in its register (thanks to the fields $\downarrow r$ provided in the description of the reception). Here again the reconfiguration of the network is done by assuming that for any broadcast, an entity might not receive it. As always in this document, we denote by $\Rightarrow^*$ the reflexive and transitive closure of the relation $\Rightarrow$.

8.4.1.3 *Example*

On Figure 8.8, we have represented a Register Broadcast Protocol whose role is to extract a list structure from a configuration. In this protocol, each entity is equipped with two registers and we assume that the value store in the first register of each entity in the initial configuration corresponds to its identity and it will never be overwritten. The idea is that when we will have a node in q_T, then there is a node either in state q_Z or in state q_H which has in its second register the identity of the node in q_T. Similarly for each process p in state q_Z there is a node either in state q_Z or in state q_H which has in its second register the identity of p. This way the protocol can build separate linked lists whose head is a process in state q_H, their tail is a node in state q_T, intermediate nodes are in state q_Z and each node has a pointer to the successor in its second register. Hence even if we do not have a communication graph in this model, we can extract a graph structure using registers as pointers.

On Figure 8.9, we have depicted an example of runs in the transition system $DRBN(BP)$ induced by the Register Broadcast Protocol BP of Figure 8.9. This run shows how this protocol can extract a linked list from an initial configuration. Note that one key feature of this protocol which ensures its correctness in term of list construction is that the communication

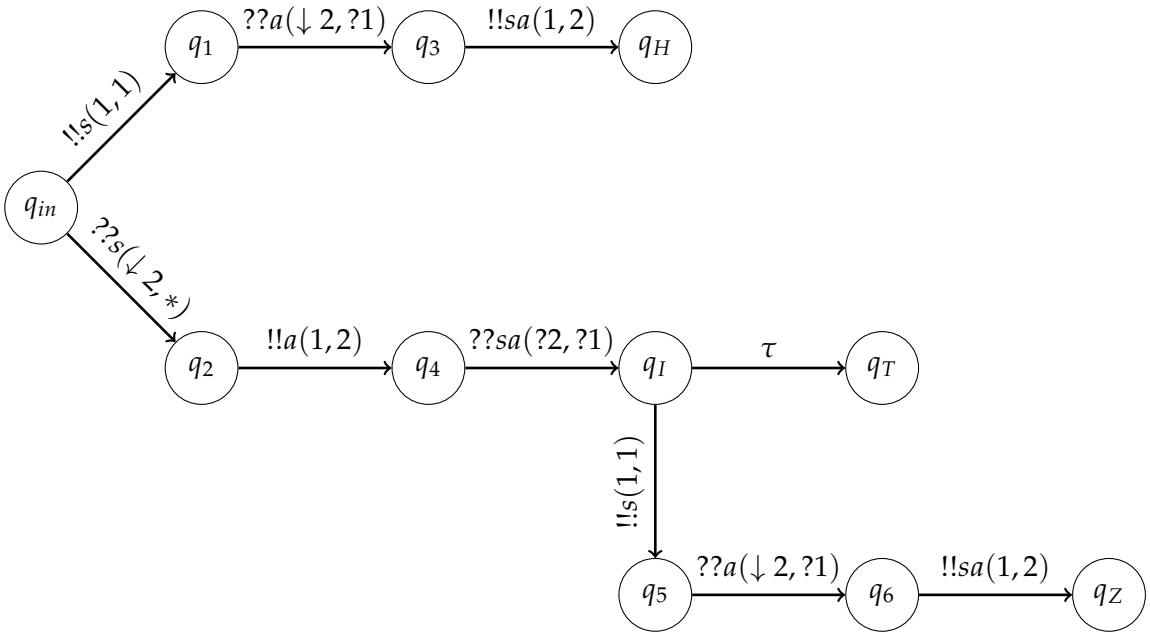

Figure 8.8: A $(2,2)$-Register Broadcast Protocol

are performed in an handshake fashion and uses the register of the processes to check the handshake. Indeed when a process p in q_{in} broadcasts a message $s(1,1)$, it could be received by more than one node, however the first one that will send a message $a(1,2)$ received by p will be consider as a successor and p will not be able to receive the other messages $a(1,2)$ sent by other nodes. Finally thanks to the message $sa(1,2)$, an acknowledgment is concluded between p and its successor which checks on reception with $??sa(?2,?1)$ that the two processes have performed an handshake. This allows to avoid interferences between processes and leads to the construction of separated linked lists. In the last configuration of the run depicted on Figure 8.8, we see a possible list, where the node with identity 1 is the head and has as successor process with identity 2, which has as successor the process with identity 4 and finally the tail of the list is the process with identity 4. Note that with more processes in the initial configuration we could have extracted a longer list.

8.4.2 *Parameterised coverability problem*

The problem we study over Data Reconfigurable Broadcast Networks extends the parameterised control state reachability problem to allow to deal as well with the contents of the registers. However here again we seek for a configuration to reach exhibiting a certain pattern which is defined thanks to what we call reachability queries. We consider a (R,F)-Register Broadcast Protocol $BP = (Q, M, \Delta, q_{in})$ and a denumerable set of variables V. A reachability query ϕ for BP is a formula gernerated by the following grammar:

$$\phi ::= q(v) \mid M_r(v) = M_{r'}(v') \mid M_r(v) \neq M_{r'}(v') \mid \phi \wedge \phi$$

where $v, v' \in V$, $q \in Q$ and $r, r' \in [1, R]$. Intuitively the variables of V will be mapped to processes of a configuration and with such formula we can state whether there is a node in a certain state and wether some registers share or not the same value. Given a configuration

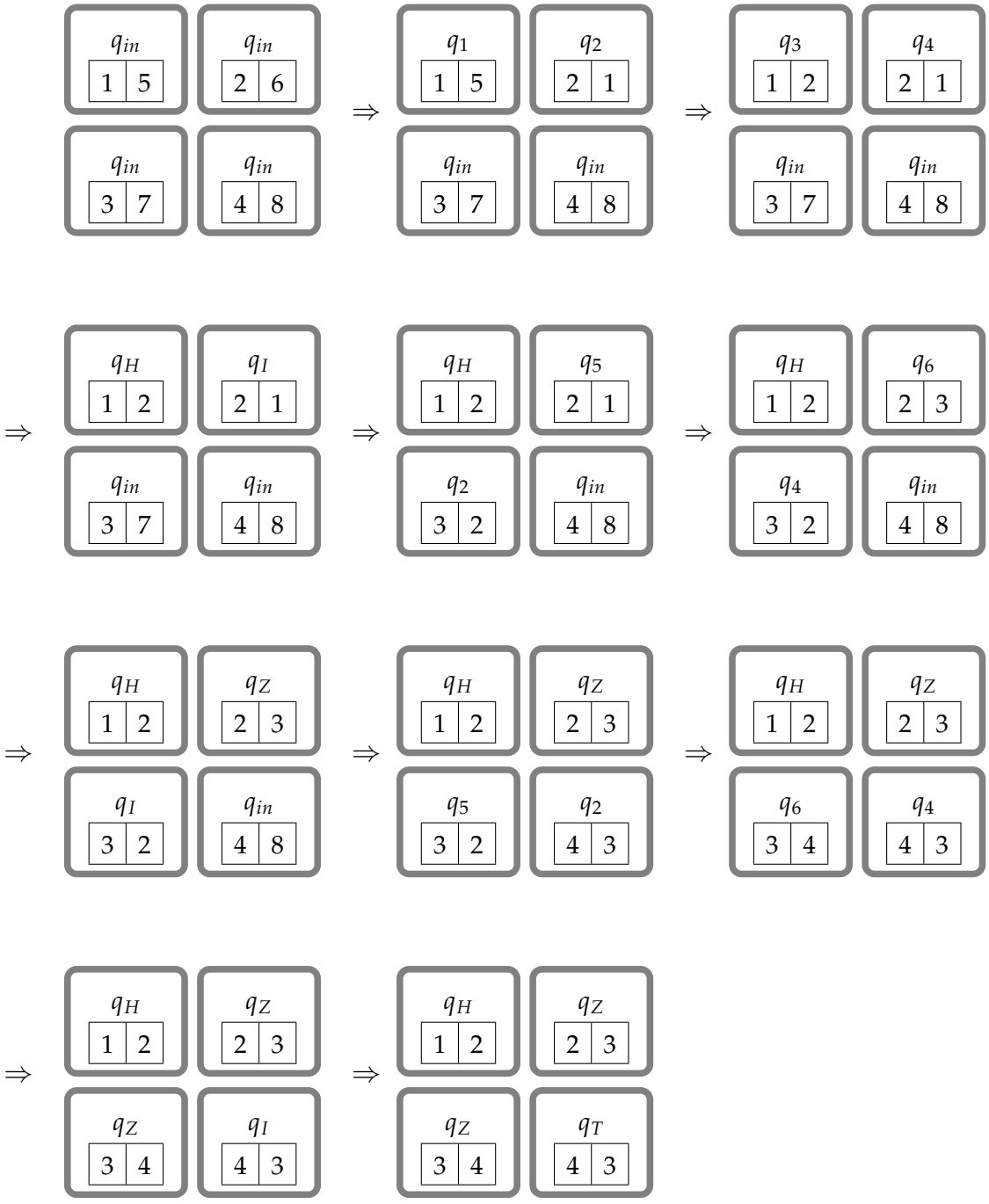

Figure 8.9: A run in the Data Reconfigurable Broadcast Network for the protocol of Figure 8.8

$\zeta = (\lambda, \mathbf{M})$ of the $DRBN(BP)$, a valuation function is a function $f : V \mapsto [1, \|\zeta\|]$ and we define the satisfaction relation for reachability queries, parameterised by a valuation function f, inductively as follows:

$$\begin{aligned}
\zeta \models_f q(v) &\overset{\text{def}}{\Leftrightarrow} \lambda[f(v)] = q \\
\zeta \models_f M_r(v) = M_{r'}(v') &\overset{\text{def}}{\Leftrightarrow} \mathbf{M}[f(v)][r] = \mathbf{M}[f(v')][r'] \\
\zeta \models_f M_r(v) \neq M_{r'}(v') &\overset{\text{def}}{\Leftrightarrow} \mathbf{M}[f(v)][r] \neq \mathbf{M}[f(v')][r'] \\
\zeta \models_f \phi \wedge \phi' &\overset{\text{def}}{\Leftrightarrow} \zeta \models_f \phi \text{ and } \zeta \models_f \phi'
\end{aligned}$$

Finally, we say that a configuration ζ satisfies a reachability query ϕ, denoted by $\zeta \models \phi$, if and only if there exists a valuation function f such that $\zeta \models_f \phi$. We have now all the elements to provide the definition of the parameterised coverability problem for Data Reconfigurable Broadcast Networks.

DRBN-Cover

Input: A Register Broadcast Protocol $BP = (Q, M, \Delta, q_{in})$ and a reachability query ϕ for BP;

Question: Does there exists an initial configuration ζ_{in} and a configuration ζ such that $\zeta_{in} \Rightarrow^* \zeta$ in $DRBN(BP)$ and $\zeta \models \phi$?

Example 8.8. *If we consider the Register Broadcast Protocol BP described on Figure 8.8, thanks to the execution in $DRBN(BP)$ shown on Figure 8.9, we know that there exists an initial configuration ζ_{in} and a configuration ζ such that $\zeta_{in} \Rightarrow^* \zeta$ and $\zeta \models q_H(v) \wedge q_Z(v') \wedge M_2(v) = M_1(v')$. On the other hand we can show that there does not exist a configuration ζ' reachable from an initial configuration such that $\zeta \models q_H(v) \wedge q_Z(v') \wedge q_Z(v'') \wedge M_2(v) = M_1(v') \wedge M_2(v) = M_1(v'')$.*

8.4.3 *Results*

The example provided in section 8.4.1.3 shows us that even if in Data Reconfigurable Broadcast Networks the communication topology does not matter and hence cannot be used as a structure (as it is the case when the topology is fixed), it is possible to use the registers of the processes as data field to store pointer towards other entities. Indeed we have seen that this latter Register Broadcast Protocol allows us to extract a list structure from an initial configuration. Reusing an idea of the proof of undecidability for the parameterised control state reachability problem Broadcast Network (without reconfiguration), see Theorem 7.1, we know that we can then use such structures to simulate a deterministic Minsky machine as explained in Section 7.2. Hence for what concerns, the parameterised coverability problem over Data Reconfigurable Broadcast Networks, our first result is negative.

Theorem 8.6. *[DST13; DST16]* DRBN-Cover *is undecidable for (R, F)-Register Broadcast Protocols with $R \geq 2$ and $F \geq 2$.*

Note that in the statement of the previous theorem, we insist on the fact that the considered protocol should be executed by entities with at least two registers and the broadcast messages

should be able to carry two data values , it is indeed necessary to extract a list structure (one register stores the identity of the node and the other one the identity of the successor of the node) and as we shall explain now we have proven in [DST13; DST16] that restricting to protocols with a single data field in the messages leads to the decidability of DRBN-COVER.

We consider now a $(R, 1)$-Register Broadcast Protocol $BP = (Q, M, \Delta, q_{in})$ and we will provide a method to analyze the behavior of the associated Data Reconfigurable Broadcast Network. As for Timed Ad Hoc Networks with a single clock and restricted configurations (see Section 7.4.3) or for Reconfigurable Broadcast Networks when considering reachability of a counting constraint (see Section 8.1.3), we rely here again on symbolic configurations which allow to reason in a finite matter over the executions of $DRBN(BP)$. The idea behind these symbolic configurations is to use graphs that keep track of the control states that may appear during an execution and as well of relations between values present in the registers.

We now provide a formal definition. A symbolic configuration for BP is a labelled graph (W, E, L) where W is a finite set of vertices, $E \subseteq W \times [1, R] \times [1, R] \times W$ is the set of labelled edges and $L : W \mapsto Q$ is a labelling function associating a control state to each vertex, such that for all $w, w' \in W$, if $w \neq w'$ then $L(w) \neq L(w')$ (i.e. there cannot be two vertices labelled with the same control state). We now explain what represent such symbolic configurations: a concrete configuration $\zeta = (\lambda, \mathbf{M})$ belongs to the concretisation $[\![(W, E, L)]\!]$ of a symbolic configuration (W, E, L) if for any process $p \in [1, \|\zeta\|]$, there is a vertex labelled with $\lambda[p]$, the control state of p, and furthermore, for any pair of processes $p, p' \in [1, \|\zeta\|]$ containing the same value in registers r and r' respectively, there exists two nodes w and w' such that $L(w) = \lambda[p]$ and $L(w') = \lambda[p']$ and $\{(w, r, r'w'), (w', r', r, w)\} \cap E \neq \emptyset$, i.e. there is an edge between w and w' labelled with the pair (r, r').

We remark then that if we take the symbolic configuration $(\{w_{in}\}, \emptyset, L_{in})$ such that $L_{in}(w_{in}) = q_{in}$ then it corresponds to the set of initial concrete configurations.

To decide DRBN-COVER over $(R, 1)$-Data Register Protocols, we propose a saturation method over symbolic configurations: it consists in applying iteratively to a symbolic configuration, a symbolic successor computation which adds information to the symbolic configuration and we do this iteratively until the obtained symbolic configuration is saturated, i.e. the symbolic successor computation does not have any effect. Finally, it remains to test whether the obtained symbolic configuration satisfies the reachability query. We provide in [DST13; DST16] the detail of this saturation method and show it is sound and complete for DRBN-COVER over $(R, 1)$-Data Register Protocol. One of the reason for this method to work is connected to the fact that in reconfigurable networks if a configuration exhibiting a specific pattern is reachable it is possible to add this pattern to all reachable configurations, since we can assume that first we reach the pattern with a subset of vertices and then we assume that these vertices do not participate in the execution to reach the rest of the configuration.

Example 8.9. *On Figure 8.10, we illustrate the key idea behind the computation of symbolic successor of a symbolic configuration. We consider the symbolic configuration on the left side of the Figure with two vertices labelled respectively with q_{in} and q_1. This configuration represents any configuration with any number of processes in q_{in} and in q_1 and where processes in state q_{in} do not share any data value whereas processes in state q_1 may have the same value in their second register that may be as well equal to the value of the first register on a process in q_{in}.*

We now give examples of edges of a $(2, 1)$-Register Broadcast Protocol BP which would lead to the construction of the symbolic configuration on the right side of the figure. Assume that we have two edges $(q_{in}, !!m(1), q_{in})$ and $(q_{in}, ??m(\downarrow 1), q_{in})$ in BP. This pair of edges indicate that some nodes in

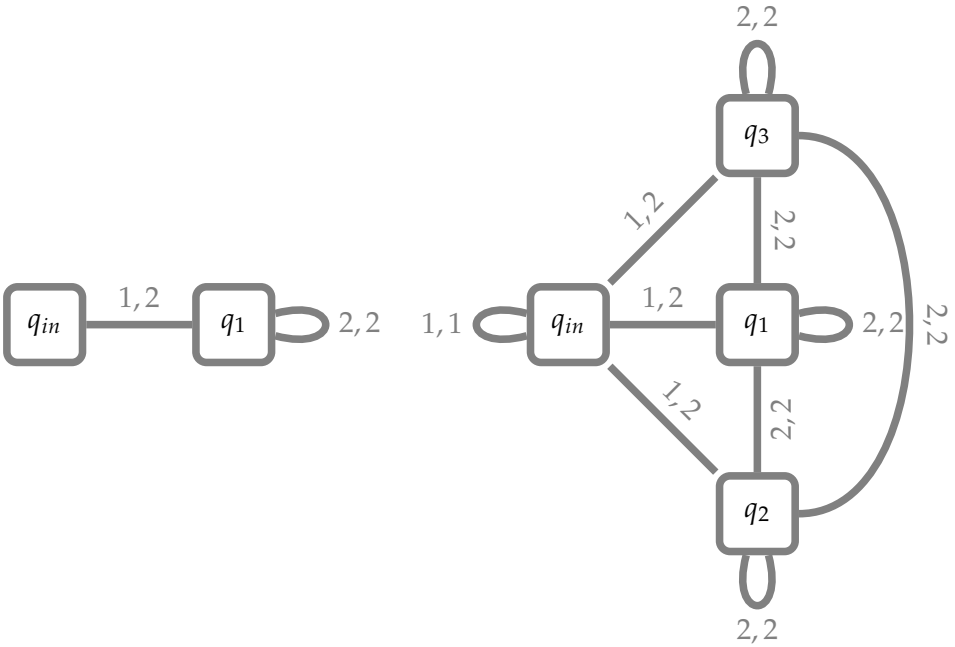

Figure 8.10: A symbolic configuration and a possible symbolic successor

control state q_{in} may now have the same value in their first register, and this leads to adding the self loop labelled by $(1,1)$ around the vertex with the label q_{in}. Consider now that we have the reception edge $(q_1, ??m(?2), q_2)$. Thanks to the edge labelled with $(1,2)$ between vertices q_{in} and q_1, we know that some processes in state q_1 may share the same value in their second register as the value stored in the first register of a process in state q_{in} which would broadcast the message m. To model the effect of this reception, we create hence a new vertex in the symbolic configuration with label q_2 which share the same properties in term of data value as the vertex labelled by q_1. Finally, if we have as well a reception edge of the form $(q_1, ??m(?\bar{2}), q_3)$, we know that the message m can be sent by a node in state q_{in} and received by nodes in state q_1 which do not share the same value in their first and second register, hence the guard is satisfied. However their could be another node in state q_{in} which shares the same value in its first register as the one present in the second registers of the nodes receiving m. As a consequence, we create a vertex labelled by q_3 which inherits as well of the data relation carried by vertex q_1.

Since a symbolic configuration has at most $card(Q)$ states and at most $card(Q)^2 \cdot R^2$ edges, we know that our saturation method which computes iteratively new symbolic configurations can be done in polynomial space. Furthermore testing whether the obtained symbolic configuration satisfies a reachability query can as well be done in polynomial space (in fact this test can be performed in non-deterministic linear time by guessing the vertices in the symbolic configuration that should be mapped to the free variables of the query). This allows us to deduce an upper bound for DRBN-Cover restricted to Register Broadcast Protocol where messages have a single data field. For what concerns the lower bound, we provide in [DST13; DST16] a reduction from the reachability problem for 1-safe Petri nets [CEP95].

Theorem 8.7. *[DST13; DST16]* DRBN-COVER *is* PSPCACE*-complete for* $(R, 1)$*-Register Broadcast Protocols.*

Part III

CONCLUSION AND PERSPECTIVES

9 CONCLUSION AND PERSPECTIVES

9.1 CONCLUSION

In this thesis I have presented my contributions for the automatic verification of computing systems with an infinite state space focusing on two families of systems, namely counter systems and networks with a parameterised number of entities communicating thanks to broadcast communication. In both cases, I have tried to establish the characteristics of the models which render the verification process complex, in other words to establish the frontier between decidability and undecidability, simple reachability questions being undecidable in both cases for the general model (see Theorem 3.1 and 7.1) and in the cases I could prove that the considered problems were decidable, I have tried to state precisely to which complexity classes they belong. In most of the cases, I have succeeded in providing lower and upper bounds and the presented works offer as a matter of fact an overview of what are the aspects of the considered infinite state systems that ease or complicate the automatic verification. More importantly, most of the results rely on new techniques or combinations of already known techniques in a new way and can be of some use to analyze other models.

Finally, I would like to insist that all the results I have obtained in the last fourteen years were the fruit of collaborations with other researchers and some of them have been produced in the context of a PhD or an internship of students and even if I presented them in my thesis, they are the consequences of team works in which each of the coauthors have participated to the research. During the period covered by this document, I have hence not only learned a lot from the scientific point of view, but as well from a social point of view on how to collaborate, to work together on a project and to supervise students.

9.2 PERSPECTIVES

I present in this section some directions for my future research, first by stating some open questions directly connected to the works I have presented in this document and second by presenting wider research thematics.

9.2.1 *Problems directly connected to the presented works*

In the first part of this thesis, we present some results concerning the model-checking of counter systems with a specific focus on flat counter systems. The starting point of all these works was when I asked Stéphane Demri to study the model-checking problem of the temporal logic freeze LTL, he had introduced in [DLN07], over counter systems. We quickly showed that in general this problem is undecidable (see Theorem 5.1) but that decidability can be regained in particular if one considered flat counter systems (see Theorem 5.2). However, we did not obtain a precise complexity characterization of this problem and we started by first looking at the results we could obtain for the linear time temporal logic without the freeze quantifier, which gave rise to all the results presented in Chapter 4.

However, the complexity of the model-checking of LTL with the freeze quantifier over flat counter systems is still unknown, but certainly some of the techniques proposed in Chapter 4 can be used to find an answer.

For what concerns the branching time temporal logics, we have shown that the model-chekcing problem for the temporal logic CTL* over flat translating counter system is equivalent (in terms of complexity) to the satisfiability problem for Presburger arithmetic (see Theorems 6.1 and 6.2) but we did not provide further results for other branching time specifications. Indeed, it is an open problem whether the model-checking of the modal μ-calculus (see Section 6.2) is decidable or not for flat Affine Counter Systems (either translating or with the finite monoid property). Whereas I believe this latter problem should be decidable, I as well think that to obtain this result, one needs to use different techniques than the one introduced for the model-checking of CTL*.

In the part dedicated to the verification of parameterised networks, we have presented that for parameterised ad hoc networks, where the communication topology remains the same along an execution, even simple reachability properties are undecidable (see Theorem 7.1) but if we allow the topology of the network to change in a complete non-deterministic fashion, then these problems become decidable and even easy (see Theorem 8.1 where we show that the reachability of a control state can be solved in polynomial time). However, from a modeling point of view, this change of the topology seems a too strong relaxation. It would be interesting to constraint the way the communication topologies can change, as in fact it has been done in [BBM18] where the authors propose to bound the number of reconfigured linked between two communication steps. Other formalisms can be taken into account in order to constraint these changes of topologies, for instance one could use some formal languages over graphs which provide allowed sequences of communication graphs, or it could as well be interesting to assume that the reconfiguration is uncontrollable and to consider in this context a turn based two player game where the first player resolves the non-determinism at the protocol level and the second player is in charge to choose who can receive the broadcast messages. Another solution could be to use a logical formalism describing the allowed movement. Studying the impact of the modelling of the reconfiguration on the verification of parameterised networks with broadcast communication is hence a wide research area.

In the models I have considered for parameterised networks, I was focusing on 'low-level' communication primitives such as message passing, rendez-vous or broadcast communication, however it would be interesting to study such networks equipped with 'high-level' storage mechanisms. For instance, a starting idea would be to assume that the entities in the networks share a set of counter variables (as the ones used in counter systems) on which they can perform operations such as test or updates. In such a model we would assume that the operations on the counters are atomic and are performed correctly. This allows to consider more high-level properties when modelling a distributed system equipped with a shared storage mechanism (one could think for instance at operation on a bank account, or at some distributed systems in charge of measuring some discrete events). From the theoretical point of view, we have seen that in some cases, looking at parameterised networks instead of networks with a fixed number of entities allows to ease the verification process in terms of complexity. Furthermore for the case where the shared storage mechanisms are counters, this would allow me to reuse an expertise I have gained both in the field of analysis of counter systems and of parameterised networks. We have begun with Julien Roupin, who

did an internship under my supervision in 2021, to study this class of models and we have obtained interesting results, but there is still work to do.

9.2.2 *Long term research perspectives*

9.2.2.1 *On the scheduling of distributed algorithms*

The basic building blocks of distributed applications are distributed algorithms, which address particular problems in specific contexts like for instance, consensus[FLP85; GR07] and its relaxation like set agreement and k-set agreement[Cha93] which ask a system with machine replication to agree on a value or a set of values. Another example of distributed task is the renaming [CRR11], which is used to provide different entities in a network with a small set of names. Even though most distributed algorithms to solve classical distributed problems are not very long and essentially consist of an iterated loop, their behavior is difficult to understand due to the numerous possible interleavings of an execution. As a result, correctness proofs of distributed algorithms are extremely intricate. Furthermore their correctness proofs strongly depend on the specific assumptions made on the considered *execution context* (usually referred to as *system model* in the realm of distributed algorithms). For example, communication can be synchronous or asynchronous, the entities in the network may or may not be equipped with a unique identifier, or a certain number of errors and failures can happen during an execution. These assumptions are crucial and may lead to different families of algorithms designed for a specific context. This is, for instance, the case for algorithms that solve the fundamental set-agreement task, in which processes collectively choose a small subset of values from a larger set of proposals (see, e.g., [Ali+12]). The behavior of many distributed algorithms tend to be very sensitive to any deviation from the context it has been conceived for. In other words, distributed algorithms tend to be *not robust*.

In the distributed computing literature, many execution frameworks have been proposed. Some are very intuitive, like the synchronous context, in which all entities behave synchronously, or the completely asynchronous context, in which arbitrary long communication delays may occur. In practice, systems are neither synchronous nor fully asynchronous. Many other models lying between the synchronous and the asynchronous model have thus been defined. Investigation of such *partially* synchronous models have also been motivated by the fact that tolerating at the same time asynchrony and failures is hard or even impossible depending on the problem being considered. Hence, one is interested in execution contexts that *degrade* (or *relax*) to some extend the assumptions of the synchronous model in order to capture real systems executions while leaving the problem under study solvable. For example, in the round-based *heard-of model* [CS09], a message in transit that is not received within a round is considered to be lost and processes then react according to the set of messages they have received in the previous round. In the Θ-*model* [WS09], a bound Θ represents the ratio between the shortest and longest time delay of messages in transit simultaneously. Contexts with partial synchrony have also been studied in the case of shared-memory systems, (see for instance [Agu+12]): one can restrict the way the different processes access the memory by, for example, bounding the number of times a process can perform read/write actions while other active processes do not access the memory.

As distributed algorithms correspond to solution to distributed problems under an execution context. In order to have a formal methodology for the design of distributed

algorithms through the lens of such contexts it is necessary to establish ways to define them. There are indeed a whole family of execution contexts such as the synchronous model, the asynchronous model, model with a certain number of failures, model with partial synchrony, etc, but there is no systematic way to define them and most of the different options are not compared to each other. There is indeed no exhaustive study that relates the different relaxations. One could want to check, for example, whether one relaxation is included in another, or, conversely, whether it allows for strictly more behaviors. It is hence necessary to perform such a study to list and classify the different execution contexts and eventually propose new ones. Furthermore, in the context of verification of concurrent systems, in order to regain decidability or to be more efficient, researchers have propose too some ways to restrict the set of considered executions by imposing some restrictions on it (for instance by bounding the number of phase in message passing system, a phase being a period of time where only one process is allowed to send messages). Such restrictions can as well be seen as specific executive contexts. It is hence as well important to understand the relation between the execution contexts developed in the distributed computing community and the restrictions used in the verification of distributed systems to under-approximate the behaviors of systems. I know describe four possible directions for this research thematic.

Design methods to synthesize distributed algorithms from a distributed problem and a scheduler. In [Del+19], we have proposed a first way to define executive contexts for distributed systems with shared memory using what we call a scheduler. The scheduler defines at each instant which process can access the shared memory and it furthermore says which process can be active infinitely often. We have indeed observed that this last point is crucial if one wants to characterise execution contexts like the obstruction-free model where every process that eventually executes in isolation has to terminate[HLM03]. This leads us to see a scheduler as a finite state Büchi automaton and we furthermore checked that such a formalism allows to represent many important execution contexts like the wait-free model, which requires that each process terminates after a finite number of its own steps[Her91], or the round-robin model, or model with a fixed number of failures. We then considered the following synthesis question: given a distributed problem (like for instance the consensus problem) and a scheduler specifying an execution context, does there exists an algorithm solving the given problem under the scheduler. In [Del+19], we have proposed a first method based on SMT solver to solve this question for some specific distributed problems in the case where the size of the distributed algorithms (corresponding to its number of states and to the local memory it uses) is fixed. Our method was able to generate automatically some new distributed algorithms (which were correct by construction), however the considered conditions were very restrictive. Hence the first task of this research direction will consist in studying deeper this synthesis question, and see in which case it is decidable (the question in its full generality in certainly undecidable, using the fact that one cannot decide given a distributed problem whether there exists a wait-free algorithm to solve it [FRT11]). I plan as well to look at schedulers defined by more complex formalisms than Büchi automata.

Build adaptive distributed algorithms. I plan in here to study in which manner it is possible to build distributed algorithms which are robust to changes of the considered schedulers. It can be explained as follows. Assume for instance that for a distributed problem, we have a set of distributed algorithms each one corresponding to a specific scheduler. The question we want to study is in which manner we can build from these different distributed algorithms, a new one which can adapt to the changes of schedulers. For this task, we will first look for reasonable formalisms to specify how the scheduling

policy can change and then we will try to see in which cases, the different algorithms can be used in order to obtain an adaptive robust algorithm. This task is especially important because it will allow to develop new algorithms which will be, in a certain sense, aware of their environment. It could as well be that in order to obtain an adaptive algorithm, we could relax dynamically the considered distributed problems. For instance, k-set agreement can be seen as a relaxation of the consensus problem (in the former the entities can decide at most k different values, whereas in the latter they all have to decide the same value). Of course, in this task, I will have to take into account what it is reasonable to monitor during the execution of the algorithms in order to detect a change in the scheduler.

Build schedulers from distributed algorithms. Another interested aspect is to study how distributed algorithms can be scheduled and in particular how to build automatically good schedulers for distributed algorithms. Given a distributed algorithm to be run in a distributed system, we plan to seek some ways to schedule the different entities in order to obtain an execution or a set of executions solving the original problem. Of course, if the algorithm is correct, we know that such a scheduler exists, and it consists in following a path in the transition system corresponding to the behaviour of the distributed algorithm. But to find such a path, the scheduler has to keep in mind the information about the current state of the system, and as a matter of fact, such a scheduler could be exponentially big. In some cases, a much smaller scheduler can however be obtained using some structural specificities of the algorithm. For instance, if one as a wait-free distributed algorithm then all possible interleavings will lead to a correct solution and a scheduler with only one state from which any process is allowed to take the hand will lead to a correct set of executions. The goal of this track is hence to provide algorithmic solutions to find good schedulers for (partially) correct distributed algorithms. To achieve this, it will be necessary to define some measures to compare schedulers and to identify the key aspects which make a distributed algorithm under a certain scheduler better than an other one. I also plan to study schedulers where concurrent actions are allowed and as well probabilistic schedulers.

Measure the concurrency of distributed algorithms by looking at their schedulers. This direction is certainly the most challenging one and it consists in proposing ways to measure the concurrency of a distributed algorithm. In fact, for a specific distributed problem there might be different algorithmic solutions and if the tasks involve many processes, it is capital for matters of efficiency to understand whether one solution will make a better use of the concurrent setting than another one. However defining such a measure is tedious. One can note that the formalisms and methods developed in this context could be well suited in the context of concurrent systems.

9.2.2.2 *Quantitative analysis of parameterised networks*

In the presented works on parameterised networks, the explored verification questions were considering mostly reachability or repeated reachability problems without taking into account any information one could get on the number of processes necessary to verify the desired property or whether such a number could be optimised. In other words, the analysis performed on the models was qualitative but actually it could be interesting to develop techniques to get quantitative information about these systems which will allow to understand how the number of processes influence the behaviour of the system. Towards this direction, population protocols, which have been recently studied from the verification point of view [Esp+17] are an interesting example. In fact, population protocols can be seen

as parameterised model where the communication is performed by pairwise rendez-vous. They furthermore respect the following property: when the population protocol ran by all the entities in the system is well-specified, eventually all the entities will be either in an accepting state or a rejecting state. In [Ang+04], Angluin et al. show that population protocols compute exactly semilinear predicates, which in our context can be stated as follows, the number (and types) of processes from which the entities eventually all ends in an accepting state is a semi-linear set and can consequently be represented by a formula of the Presburger arithmetic (first order logic over the naturals with addition). Consequently, for such systems, it is possible to represent finitely the numbers for which the protocols terminate and furthermore to analyse some properties of these numbers. The purpose of this aspect of my research project consists in developing techniques to perform quantitative analysis of parameterised networks, to understand what are the mathematical tools needed for this purpose and what are the connections with existing works on infinite state systems. I propose here again some possible research directions.

Quantitative analysis on the number of participants in parameterised systems. First, I will track properties concerning the number of initial processes which allow to guarantee a certain property. One interesting aspect is to check whether the considered model has a cutoff property, i.e. whether there exists a bound b such that if the model has a problem for a certain number of processes then this problem occurs as well with b processes. In fact, when a model has a cutoff, the verification of the systems boils down to the verification of a finite state system. In [Ami+14], the authors provide some classes of systems communicating through rendez-vous for which there exists indeed a cutoff for some properties expressed in linear temporal logic. However in many important cases such a cutoff property on the model is generally not verified. For instance, in a system where processes communicate through pairwise rendez-vous and the goal is that there is an execution for which all the processes end in a final state, it is easy to build an example for which the goal is satisfied if and only if the number of entities is even. However, instead of trying to see if a family of systems respects a certain cutoff property, it could be nice to have algorithms which given a system (and a specification) decides whether it has a cutoff. Even better, if we could find a logical formalism (like for instance formulae of the Presburger arithmetic) to characterise for a given system the number of processes which ensures a certain property by then it would be possible to reason on formulae to decide the existence of a cutoff. Consequently, one of my first objective will be to seek for families of parameterised systems for which one can design algorithms to reason on the number of entities which ensure a certain property. In [HS20], we have obtained first results in this direction providing an algorithm to decide whether there is a cutoff for systems with rendez-vous communication and with a specification which requires all the entities to reach a final state. I plan furthermore to deal with quantitative aspects in the considered properties: for instance, it could be nice to state properties for a protocol about the proportion between the number of entities which will always verify a certain properties and the one which will not but whose presence is useful to the other ones. For example, it can be the case that in a system all the entities do not succeed in reaching a final state but a majority of entities always succeed.

Quantitative analysis of parameterised systems with costs. Another aspect to go beyond the properties concerning only the number of participants in the networks would be to insert some cost or rewards in the models and then to see if it is possible for instance to compute functions or relations taking into account the number of participants and connecting it with these costs. For instance, assume we have a model with synchronous

messages and non-determinism and we want to establish the smallest number of concurrent steps (knowing that in one concurrent step, many processes can evolve) which allows all the entities to reach a final state. Is it possible to find a function which associates to each number of participants this number of steps? If we do not consider anymore the minimal number of steps, is there a definable relation which links the number of entities with such a cost? To determine if such functions/relations are definable and can be computed will allow to see more in details the computational power of the considered systems and to understand better what can be expected when such models are upgraded with new functionalities so that they can represent more faithfully existing algorithms. This works represents a real challenge for formal methods. In fact, it is well known that tackling quantitative aspects is more difficult than solving qualitative problems especially when the considered systems have an unbounded number of states. And it is even more difficult to conceive algorithmic methods which automatically compute functions or relations.

Analysis of the reachability set of the Petri nets corresponding to parameterised networks. The last point I could pursue in this part of my project is more exploratory. As one can notice in the paper of German and Sistla [GS92] but as well in the more recent works on population protocols [Esp+17], parameterised networks with simple protocols (represented for instance by a finite state automaton) and simple communication paradigm (as for instance rendez-vous or synchronous) can be encoded into Petri nets and verification questions on such systems can be automatically deduced from the decidability of problems like coverability or reachability in Petri nets. However for what concerns qualitative aspects, such a methodology can be hard to apply. To illustrate this idea, if we consider the population protocols, it has been shown that they compute semilinear predicates and hence in a certain matter they can be encoded into a Petri net whose reachability set (or part of it) is semi-linear. It is known that some Petri nets have a semilinear reachability sets but other not[HP79] and there exist some subclasses of Petri nets with a semilinear reachability set, the most famous one being the cyclic Petri net[AK77]. However, it seems that the Petri net built from a population protocol does not belong to any of these subclasses (otherwise such a reduction would simplify the proofs presented in [Ang+04] or [Esp+17]). On the other hand, it is commonly accepted that the problem of knowing if a given Petri net has a semilinear reachability set is decidable and it has been proved in Hauschildt's PhD thesis [Hau90], but the results of this thesis have never been published and the content is very technical and difficult to read. My aim is hence to understand if there are new classes of Petri nets with a semilinear reachability set, focusing on Petri nets characterising some parameterised systems. I plan as well to see if some hypothesis on the considered systems render the results of Hauschildt more easy to parse in order to use its algorithm to build the corresponding semilinear sets. Even if this task seems very challenging (Hauschildt's result is known from more than 20 years, but nobody since then tried to simplify it), the period seems adequate for an exploration on this direction thanks to the new works that have made the algorithm for the reachability problem in Petri nets more accessible [LS15] knowing that part of the algorithm of Hauschildt is based on the decomposition used in the reachability algorithm.

BIBLIOGRAPHY

[Abd+96] P. A. Abdulla, K. Cerans, B. Jonsson, and Y.-K. Tsay. "General Decidability Theorems for Infinite-State Systems." In: *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science - LICS'96*. IEEE Computer Society, 1996, pp. 313–321.

[Abd+00] P. A. Abdulla, K. Cerans, B. Jonsson, and Y.-K. Tsay. "Algorithmic Analysis of Programs with Well Quasi-ordered Domains." In: *Information and Computation* 160.1-2 (2000), pp. 109–127.

[Abd+16a] P. A. Abdulla, R. Ciobanu, R. Mayr, A. Sangnier, and J. Sproston. "Qualitative Analysis of VASS-Induced MDPs." In: *Proceedings of the 19th International Conference on Foundations of Software Science and Computation Structures - FOSSACS'16*. Vol. 9634. Lecture Notes in Computer Science. Springer, 2016, pp. 319–334.

[Abd+11] P. A. Abdulla, G. Delzanno, O. Rezine, A. Sangnier, and R. Traverso. "On the Verification of Timed Ad Hoc Networks." In: *Proceedings of the 9th International Conference on Formal Modeling and Analysis of Timed Systems - FORMATS'11*. Vol. 6919. Lecture Notes in Computer Science. Springer, 2011, pp. 256–270.

[Abd+16b] P. A. Abdulla, G. Delzanno, O. Rezine, A. Sangnier, and R. Traverso. "Parameterized verification of time-sensitive models of ad hoc network protocols." In: *Theoretical Computer Science* 612 (2016), pp. 1–22.

[AHM07] P. A. Abdulla, N. Ben Henda, and R. Mayr. "Decisive Markov Chains." In: *Logical Methods in Computer Science* 3.4 (2007).

[AJ96] P. A. Abdulla and B. Jonsson. "Verifying Programs with Unreliable Channels." In: *Information and Computation* 127.2 (1996), pp. 91–101.

[AJ03] P. A. Abdulla and B. Jonsson. "Model checking of systems with many identical timed processes." In: *Theoretical Computer Science* 290.1 (2003), pp. 241–264.

[Abd+13] P. A. Abdulla, R. Mayr, A. Sangnier, and J. Sproston. "Solving Parity Games on Integer Vectors." In: *Proceedings of the 24th International Conference on Concurrency Theory - CONCUR'13*. Vol. 8052. Lecture Notes in Computer Science. Springer, 2013, pp. 106–120.

[Agu+12] M. K. Aguilera, C. Delporte-Gallet, H. Fauconnier, and S. Toueg. "Partial synchrony based on set timeliness." In: *Distributed Computing* 25.3 (2012), pp. 249–260.

[Ali+12] D. Alistarh, S. Gilbert, R. Guerraoui, and C. Travers. "Of Choices, Failures and Asynchrony: The Many Faces of Set Agreement." In: *Algorithmica* 62.1-2 (2012), pp. 595–629.

[AD94] R. Alur and D. L. Dill. "A Theory of Timed Automata." In: *Theor. Comput. Sci.* 126.2 (1994), pp. 183–235.

[AH89] R. Alur and T. A. Henzinger. "A Really Temporal Logic." In: *30th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, 1989, pp. 164–169.

[Ami+14] B. Aminof, S. Jacobs, A. Khalimov, and S. Rubin. "Parameterized Model Checking of Token-Passing Systems." In: *Proceedings of the 15th International Conference on Verification, Model Checking, and Abstract Interpretation - VMCAI'14*. Vol. 8318. Lecture Notes in Computer Science. Springer, 2014, pp. 262–281.

[ARZ15] B. Aminof, S. Rubin, and F. Zuleger. "On the Expressive Power of Communication Primitives in Parameterised Systems." In: *Proceedings of the 20th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning - LPAR-20'15*. Vol. 9450. Lecture Notes in Computer Science. Springer, 2015, pp. 313–328.

[Ang+04] D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, and R. Peralta. "Computation in networks of passively mobile finite-state sensors." In: *Proceedings of the 23rd Annual ACM Symposium on Principles of Distributed Computing - PODC'04*. ACM, 2004, pp. 290–299.

[AK77] T. Araki and T. Kasami. "Decidable Problems on the Strong Connectivity of Petri Net Reachability Sets." In: *Theoretical Computer Science* 4.1 (1977), pp. 99–119.

[AB09] S. Arora and B. Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009. ISBN: 978-0-511-53381-5.

[Ati10] M. F. Atig. "Global Model Checking of Ordered Multi-Pushdown Systems." In: *Proceedings of the IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science - FSTTCS'10*. Vol. 8. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2010, pp. 216–227.

[BK08] C. Baier and J.-P. Katoen. *Principles of model checking*. MIT Press, 2008. ISBN: 978-0-262-02649-9.

[BBM18] A. R. Balasubramanian, N. Bertrand, and N. Markey. "Parameterized Verification of Synchronization in Constrained Reconfigurable Broadcast Networks." In: *Proceedings of the 24th International Conference on Tools and Algorithms for the Construction and Analysis of Systems - TACAS'18*. Vol. 10806. Lecture Notes in Computer Science. Springer, 2018, pp. 38–54.

[Ber77] L. Berman. "Precise Bounds for Presburger Arithmetic and the Reals with Addition: Preliminary Report." In: *Proceedings of the 18th Annual Symposium on Foundations of Computer Science- FOCS'77*. IEEE Computer Society, 1977, pp. 95–99.

[BBM21] N. Bertrand, P. Bouyer, and A. Majumdar. "Reconfiguration and Message Losses in Parameterized Broadcast Networks." In: *Logical Methods in Computer Science* 17.1 (2021).

[Ber+12] N. Bertrand, G. Delzanno, B. König, A. Sangnier, and J. Stückrath. "On the Decidability Status of Reachability and Coverability in Graph Transformation Systems." In: *Proceedings of 23rd International Conference on Rewriting Techniques and Applications - RTA'12*. Vol. 15. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012, pp. 101–116.

[Ber+19] N. Bertrand, M. Dewaskar, B. Genest, H. Gimbert, and A. Amit Godbole. "Controlling a population." In: *Logical Methods in Computer Science* 15.3 (2019).

[BFS14] N. Bertrand, P. Fournier, and A. Sangnier. "Playing with Probabilities in Reconfigurable Broadcast Networks." In: *Proceedings of the 17th International Conference on Foundations of Software Science and Computation Structures - FOSSACS'14*. Vol. 8412. Lecture Notes in Computer Science. Springer, 2014, pp. 134–148.

[BFS15] N. Bertrand, P. Fournier, and A. Sangnier. "Distributed Local Strategies in Broadcast Networks." In: *Proceedings of the 26th International Conference on Concurrency Theory - CONCUR'15*. Vol. 42. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015, pp. 44–57.

[BB07] H. Björklund and M. Bojanczyk. "Bounded Depth Data Trees." In: *Proceedings of the 34th International Colloquium on Automata, Languages and Programming - ICALP'07*. Vol. 4596. Lecture Notes in Computer Science. Springer, 2007, pp. 862–874.

[Blo+21] M. Blondin, J. Esparza, S. Jaax, and P. J. Meyer. "Towards efficient verification of population protocols." In: *Formal Methods in System Design* 57.3 (2021), pp. 305–342.

[Boi99] B. Boigelot. *Symbolic Methods for Exploring Infinite State Spaces*. PhD, Université de Liège, 1999.

[Boj+11] M. Bojanczyk, C. David, A. Muscholl, Schwentick T, and L. Segoufin. "Two-variable logic on data words." In: *ACM Trans. Comput. Log.* 12.4 (2011), 27:1–27:26.

[Boj+09] M. Bojanczyk, A. Muscholl, T. Schwentick, and L. Segoufin. "Two-variable logic on data trees and XML reasoning." In: *J. ACM* 56.3 (2009), 13:1–13:48.

[BGS14] B. Bollig, P. Gastin, and J. Schubert. "Parameterized Verification of Communicating Automata under Context Bounds." In: *Proceedings of the 8th International Workshop on Reachability Problems - RP'14*. Vol. 8762. Lecture Notes in Computer Science. Springer, 2014, pp. 45–57.

[BQS17] B. Bollig, K. Quaas, and A. Sangnier. "The Complexity of Flat Freeze LTL." In: *Proceedings of the 28th International Conference on Concurrency Theory - CONCUR'17*. Vol. 85. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 33:1–33:16.

[BQS19] B. Bollig, K. Quaas, and A. Sangnier. "The Complexity of Flat Freeze LTL." In: *Logical Methods in Computer Science* 15.3 (2019).

[BRS21] B. Bollig, F. Ryabinin, and A. Sangnier. "Reachability in Distributed Memory Automata." In: *Proceedings of the 29th EACSL Annual Conference on Computer Science Logic - CSL'21*. Vol. 183. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 13:1–13:16.

[BSS21] B. Bollig, A. Sangnier, and O. Stietel. "Local First-Order Logic with Two Data Values." In: *Proceedings of the 41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science - FSTTCS'21*. Vol. 213. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 39:1–39:15.

[BT76] I. Borosh and L. Treybig. "Bounds on positive integral solutions of linear Diophantine equations." In: *American Mathematical Society* 55 (1976), pp. 299–304.

[Bou+11] A. Bouajjani, M. Bozga, P. Habermehl, R. Iosif, P. Moro, and T. Vojnar. "Programs with lists are counter automata." In: *Formal Methods in System Design* 38.2 (2011), pp. 158–192.

[BEM97] A. Bouajjani, J. Esparza, and O. Maler. "Reachability Analysis of Pushdown Automata: Application to Model-Checking." In: *Proceedings of the 8th International Conference on Concurrency Theory - CONCUR'97*. Vol. 1243. Lecture Notes in Computer Science. Springer, 1997, pp. 135–150.

[Bou+16] P. Bouyer, N. Markey, M. Randour, A. Sangnier, and D. Stan. "Reachability in Networks of Register Protocols under Stochastic Schedulers." In: *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming - ICALP'16*. Vol. 55. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016, 106:1–106:14.

[BPT03] P. Bouyer, A. Petit, and D. Thérien. "An algebraic approach to data languages and timed languages." In: *Information and Computation* 182.2 (2003), pp. 137–162.

[BIL09] M. Bozga, R. Iosif, and Y. Lakhnech. "Flat Parametric Counter Automata." In: *Fundamenta Informaticae* 91.2 (2009), pp. 275–303.

[BZ83] D. Brand and P. Zafiropulo. "On Communicating Finite-State Machines." In: *Journal of the ACM* 30.2 (1983), pp. 323–342.

[BJK10] T. Brázdil, P. Jancar, and A. Kucera. "Reachability Games on Extended Vector Addition Systems with States." In: *Proceedings of the 37th International Colloquium on Automata, Languages and Programming (Part II) - ICALP'10*. Vol. 6199. Lecture Notes in Computer Science. Springer, 2010, pp. 478–489.

[Brá+15] T. Brázdil, S. Kiefer, A. Kucera, and P. Novotný. "Long-Run Average Behaviour of Probabilistic Vector Addition Systems." In: *Proceedings of the 30th Annual ACM/IEEE Symposium on Logic in Computer Science - LICS'15*. IEEE Computer Society, 2015, pp. 44–55.

[CRR11] A. Castañeda, S. Rajsbaum, and M. Raynal. "The renaming problem in shared memory systems: An introduction." In: *Computer Science Review* 5.3 (2011), pp. 229–251.

[CS09] B. Charron-Bost and A. Schiper. "The Heard-Of model: computing in distributed systems with benign faults." In: *Distributed Computing* 22.1 (2009), pp. 49–71.

[Cha+09] K. Chatterjee, L. de Alfaro, M. Faella, and A. Legay. "Qualitative Logics and Equivalences for Probabilistic Systems." In: *Logical Methods in Computer Science* 5.2 (2009).

[Cha+10] K. Chatterjee, L. Doyen, T. A. Henzinger, and J.-F. Raskin. "Generalized Mean-payoff and Energy Games." In: *Proceedings of the IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science - FSTTCS'10*. Vol. 8. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2010, pp. 505–516.

[CRR12] K. Chatterjee, M. Randour, and J.-F. Raskin. "Strategy Synthesis for Multi-Dimensional Quantitative Objectives." In: *Proceedings of the 23rd International Conference on Concurrency Theory - CONCUR'12*. Vol. 7454. Lecture Notes in Computer Science. Springer, 2012, pp. 115–131.

[Cha93] S. Chaudhuri. "More Choices Allow More Faults: Set Consensus Problems in Totally Asynchronous Systems." In: *Information and Computation* 105.1 (1993), pp. 132–158.

[CEP95] A. Cheng, J. Esparza, and J. Palsberg. "Complexity Results for 1-Safe Nets." In: *Theoretical Computer Science* 147.1&2 (1995), pp. 117–136.

[Chi+19] D. Chistikov, W. Czerwinski, P. Hofman, M. Pilipczuk, and M. Wehar. "Shortest paths in one-counter systems." In: *Logical Methods in Computer Science* 15.1 (2019).

[Cla+04] E. M. Clarke, M. Talupur, T. Touili, and H. Veith. "Verification by Network Decomposition." In: *Proceedings of the 15th International Conference on Concurrency Theory - CONCUR'04*. Vol. 3170. Lecture Notes in Computer Science. Springer, 2004, pp. 276–291.

[Col+17] T. Colcombet, M. Jurdzinski, R. Lazic, and S. Schmitz. "Perfect half space games." In: *Proceedings of the 32nd Annual ACM/IEEE Symposium on Logic in Computer Science - LICS'17*. IEEE Computer Society, 2017, pp. 1–11.

[CJ98] H. Comon and Y. Jurski. "Multiple Counters Automata, Safety Analysis and Presburger Arithmetic." In: *Proceedings of the 10th International Conference on Computer Aided Verification - CAV'98*. Vol. 1427. Lecture Notes in Computer Science. Springer, 1998, pp. 268–279.

[CS14] J.-B. Courtois and S. Schmitz. "Alternating Vector Addition Systems with States." In: *Proceedings of the 39th International Symposium on Mathematical Foundations of Computer Science (Part I) - MFCS'14*. Vol. 8634. Lecture Notes in Computer Science. Springer, 2014, pp. 220–231.

[Cze+19a] W. Czerwinski, L. Daviaud, N. Fijalkow, M. Jurdzinski, R. Lazic, and P. Parys. "Universal trees grow inside separating automata: Quasi-polynomial lower bounds for parity games." In: *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms - SODA'19*. SIAM, 2019, pp. 2333–2349.

[Cze+19b] W. Czerwinski, S. Lasota, R. Lazic, J. Leroux, and F. Mazowiecki. "The reachability problem for Petri nets is not elementary." In: *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing - STOC'19*. ACM, 2019, pp. 24–33.

[D'A+07] D. D'Aprile, S. Donatelli, A. Sangnier, and J. Sproston. "From Time Petri Nets to Timed Automata: An Untimed Approach." In: *Proceedings of the 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems - TACAS'07*. Vol. 4424. Lecture Notes in Computer Science. Springer, 2007, pp. 216–230.

[Dec+17] N. Decker, P. Habermehl, M. Leucker, A. Sangnier, and D. Thoma. "Model-Checking Counting Temporal Logics on Flat Structures." In: *Proceedings of the 28th International Conference on Concurrency Theory - CONCUR'17, September 5-8, 2017, Berlin, Germany*. Vol. 85. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 29:1–29:17.

[Del+19] C. Delporte-Gallet, H. Fauconnier, Y. Jurski, F. Laroussinie, and A. Sangnier. "Towards Synthesis of Distributed Algorithms with SMT Solvers." In: *Proceedings of the 7th International Conference on Networked Systems - NETYS'19*. Vol. 11704. Lecture Notes in Computer Science. Springer, 2019, pp. 200–216.

[DST13] G. Delzanno, A. Sangnier, and R. Traverso. "Parameterized Verification of Broadcast Networks of Register Automata." In: *Proceedings of th 7th International Workshop on Reachability Problems - RP'13*. Vol. 8169. Lecture Notes in Computer Science. Springer, 2013, pp. 109–121.

[DST16] G. Delzanno, A. Sangnier, and R. Traverso. "Adding Data Registers to Parameterized Networks with Broadcast." In: *Fundamenta Informaticae* 143.3-4 (2016), pp. 287–316.

[Del+12] G. Delzanno, A. Sangnier, R. Traverso, and G. Zavattaro. "On the Complexity of Parameterized Reachability in Reconfigurable Broadcast Networks." In: *Proceedings of the IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science - FSTTCS'12*. Vol. 18. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012, pp. 289–300.

[DSZ10] G. Delzanno, A. Sangnier, and G. Zavattaro. "Parameterized Verification of Ad Hoc Networks." In: *Proceedings of the 21th International Conference on Concurrency Theory - CONCUR'10*. Vol. 6269. Lecture Notes in Computer Science. Springer, 2010, pp. 313–327.

[DSZ11] G. Delzanno, A. Sangnier, and G. Zavattaro. "On the Power of Cliques in the Parameterized Verification of Ad Hoc Networks." In: *Proceedings of the 14th International Conference on Foundations of Software Science and Computational Structures - FOSSACS'11*. Vol. 6604. Lecture Notes in Computer Science. Springer, 2011, pp. 441–455.

[DSZ12] G. Delzanno, A. Sangnier, and G. Zavattaro. "Verification of Ad Hoc Networks with Node and Communication Failures." In: *Proceedings of the Joint 14th IFIP WG 6.1 International Conferenceand and 32nd IFIP WG 6.1 International Conference on Formal Techniques for Distributed Systems - FMOODS/FORTE'12*. Vol. 7273. Lecture Notes in Computer Science. Springer, 2012, pp. 235–250.

[DDS12] S. Demri, A. K. Dhar, and A. Sangnier. "Taming Past LTL and Flat Counter Systems." In: *Proceedings of the 6th International Joint Conference on Automated Reasoning - IJCAR'12*. Vol. 7364. Lecture Notes in Computer Science. Springer, 2012, pp. 179–193.

[DDS13] S. Demri, A. K. Dhar, and A. Sangnier. "On the Complexity of Verifying Regular Properties on Flat Counter Systems," in: *Proceedings of the 40th International Colloquium on Automata, Languages, and Programming Part II - ICALP'13*. Vol. 7966. Lecture Notes in Computer Science. Springer, 2013, pp. 162–173.

[DDS14] S. Demri, A. K. Dhar, and A. Sangnier. "Equivalence Between Model-Checking Flat Counter Systems and Presburger Arithmetic." In: *Proceedings of the 8th International Workshop on Reachability Problems - RP'14*. Vol. 8762. Lecture Notes in Computer Science. Springer, 2014, pp. 85–97.

[DDS15] S. Demri, A. K. Dhar, and A. Sangnier. "Taming past LTL and flat counter systems." In: *Information and Computation* 242 (2015), pp. 306–339.

[DDS18] S. Demri, A. Kumar Dhar, and A. Sangnier. "Equivalence between model-checking flat counter systems and Presburger arithmetic." In: *Theoretical Computer Science* 735 (2018), pp. 2–23.

[Dem+10] S. Demri, A. Finkel, V. Goranko, and G. van Drimmelen. "Model-checking CTL* over flat Presburger counter systems." In: *Journal of Applied Non Classical Logics* 20.4 (2010), pp. 313–344.

[DL09] S. Demri and R. Lazic. "LTL with the freeze quantifier and register automata." In: *ACM Trans. Comput. Log.* 10.3 (2009), 16:1–16:30.

[DLN07] S. Demri, R. Lazic, and D. Nowak. "On the freeze quantifier in Constraint LTL: Decidability and complexity." In: *Information and Computation* 205.1 (2007), pp. 2–24.

[DLS08] S. Demri, R. Lazic, and A. Sangnier. "Model Checking Freeze LTL over One-Counter Automata." In: *Proceedings of the 11th International Conference on Foundations of Software Science and Computational Structures - FOSSACS'08*. Vol. 4962. Lecture Notes in Computer Science. Springer, 2008, pp. 490–504.

[DLS10] S. Demri, R. Lazic, and A. Sangnier. "Model checking memoryful linear-time logics over one-counter automata." In: *Theoretical Computer Science* 411.22-24 (2010), pp. 2298–2316.

[DS10] S. Demri and A. Sangnier. "When Model-Checking Freeze LTL over Counter Machines Becomes Decidable." In: *Proceedings of the 13th international Conference on Foundations of Software Science and Computational Structure - FOSSACS'10*. Vol. 6014. Lecture Notes in Computer Science. Springer, 2010, pp. 176–190.

[Dic13] L. E. Dickson. "Finiteness of the Odd Perfect and Primitive Abundant Numbers with n Distinct Prime Factors." In: *American Journal of Mathematics* 35 (1913), p. 413.

[Din92] G. Ding. "Subgraphs and well-quasi-ordering." In: *J. Graph Theory* 16.5 (1992), pp. 489–502.

[Dur+17] A. Durand-Gasselin, J. Esparza, P. Ganty, and R. Majumdar. "Model checking parameterized asynchronous shared-memory systems." In: *Formal Methods in System Design* 50.2-3 (2017), pp. 140–167.

[EJS93] E. A. Emerson, C. S. Jutla, and A. P. Sistla. "On Model-Checking for Fragments of ¯-Calculus." In: *Proceedings of the 5th International Conference Computer Aided Verification - CAV '93*. Vol. 697. Lecture Notes in Computer Science. Springer, 1993, pp. 385–396.

[EH86] E. Allen Emerson and Joseph Y. Halpern. "Sometimes and Not Never revisited: on branching versus linear time temporal logic." In: *J. ACM* 33.1 (1986), pp. 151–178.

[Esp14] J. Esparza. "Keeping a Crowd Safe: On the Complexity of Parameterized Verification (Invited Talk)." In: *Proceedings of the 31st International Symposium on Theoretical Aspects of Computer Science - STACS'14)*. Vol. 25. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014, pp. 1–10.

[EFM99] J. Esparza, A. Finkel, and R. Mayr. "On the Verification of Broadcast Protocols." In: *Proceedings of the 14th Annual IEEE Symposium on Logic in Computer Science - LICS'99*. IEEE Computer Society, 1999, pp. 352–359.

[Esp+17] J. Esparza, P. Ganty, J. Leroux, and R. Majumdar. "Verification of population protocols." In: *Acta Informatica* 54.2 (2017), pp. 191–215.

[EGM16] J. Esparza, P. Ganty, and R. Majumdar. "Parameterized Verification of Asynchronous Shared-Memory Systems." In: *Journal of the ACM* 63.1 (2016), 10:1–10:48.

[Esp+21] J. Esparza, S. Jaax, M. A. Raskin, and C. Weil-Kennedy. "The complexity of verifying population protocols." In: *Distributed Comput.* 34.2 (2021), pp. 133–177.

[EN94] J. Esparza and M. Nielsen. "Decidability Issues for Petri Nets - a survey." In: *Bull. EATCS* 52 (1994), pp. 244–262.

[FJ15] J. Fearnley and M. Jurdzinski. "Reachability in two-clock timed automata is PSPACE-complete." In: *Information and Computation* 243 (2015), pp. 26–36.

[FL02] A. Finkel and J. Leroux. "How to Compose Presburger-Accelerations: Applications to Broadcast Protocols." In: *Proceedings of 22nd Conference on Foundations of Software Technology and Theoretical Computer Science - FSTTCS'02*. Ed. by Manindra Agrawal and Anil Seth. Vol. 2556. Lecture Notes in Computer Science. Springer, 2002, pp. 145–156.

[FLS07] A. Finkel, E. Lozes, and A. Sangnier. "Towards Model-Checking Programs with Lists." In: *Revisited Selected Papers of the Internation Conference Infinity in Logic and Computation ILC'07*. Vol. 5489. Lecture Notes in Computer Science. Springer, 2007, pp. 56–86.

[FS08] A. Finkel and A. Sangnier. "Reversal-Bounded Counter Machines Revisited." In: *Proceedings of the 33rd International Symposium on Mathematical Foundations of Computer Science - MFC''08*. Vol. 5162. Lecture Notes in Computer Science. Springer, 2008, pp. 323–334.

[FS10] A. Finkel and A. Sangnier. "Mixing Coverability and Reachability to Analyze VASS with One Zero-Test." In: *Proceedings of the 36th Conference on Current Trends in Theory and Practice of Computer Science - SOFSEM'10*. Vol. 5901. Lecture Notes in Computer Science. Springer, 2010, pp. 394–406.

[FS01] A. Finkel and P. Schnoebelen. "Well-structured transition systems everywhere!" In: *Theoretical Computer Science* 256.1-2 (2001), pp. 63–92.

[FLP85] M. J. Fischer, N. A. Lynch, and M. Paterson. "Impossibility of Distributed Consensus with One Faulty Process." In: *Journal of the ACM* 32.2 (1985), pp. 374–382.

[Fou15] P. Fournier. "Parameterized verification of networks of many identical processes. (Vérification paramétrée de réseaux composés d'une multitude de processus identiques)." PhD thesis. University of Rennes 1, France, 2015.

[FRT11] P. Fraigniaud, S. Rajsbaum, and C. Travers. "Locality and Checkability in Wait-Free Computing." In: *Proceedings of the 25th International Symposium on Distributed Computing - DISC'11*. Vol. 6950. Lecture Notes in Computer Science. Springer, 2011, pp. 333–347.

[FO97] L. Fribourg and H. Olsén. "A Decompositional Approach for Computing Least Fixed-Points of Datalog Programs with Z-Counters." In: *Constraints An International Journal* 2.3/4 (1997), pp. 305–335.

[Gal76] Z. Galil. "Hierarchies of Complete Problems." In: *Acta Informatica* 6 (1976), pp. 77–88.

[GS92] S. M. German and A. P. Sistla. "Reasoning about Systems with Many Processes." In: *Journal of the ACM* 39.3 (1992), pp. 675–735.

[Göl+10] S. Göller, C. Haase, J. Ouaknine, and J. Worrell. "Model Checking Succinct and Parametric One-Counter Automata." In: *Proceeding of 37th International Colloquium on Automata, Languages and Programming - ICALP'10*. Vol. 6199. Lecture Notes in Computer Science. Springer, 2010, pp. 575–586.

[GL13] S. Göller and M. Lohrey. "Branching-Time Model Checking of One-Counter Processes and Timed Automata." In: *SIAM J. Comput.* 42.3 (2013), pp. 884–923.

[Gor96] V. Goranko. "Hierarchies of Modal and Temporal Logics with Reference Pointers." In: *Journal of Logic, Language and Information* 5.1 (1996), pp. 1–24.

[GTW02] E. Grädel, W. Thomas, and T. Wilke, eds. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*. Vol. 2500. Lecture Notes in Computer Science. Springer, 2002.

[GR07] R. Guerraoui and M. Raynal. "The Alpha of Indulgent Consensus." In: *The Computer Journal* 50.1 (2007), pp. 53–67.

[Haa14] C. Haase. "Subclasses of presburger arithmetic and the weak EXP hierarchy." In: *Proceedings of the Joint Meeting of the 23rd EACSL Annual Conference on Computer Science Logic and the 29th Annual ACM/IEEE Symposium on Logic in Computer Science - CSL-LICS '14*. ACM, 2014, 47:1–47:10.

[Haa+09] C. Haase, S.Kreutzer, J. Ouaknine, and J. Worrell. "Reachability in Succinct and Parametric One-Counter Automata." In: *Proceedings of the 20th International Conference on Concurrency Theory - CONCUR'09*. Vol. 5710. Lecture Notes in Computer Science. Springer, 2009, pp. 369–383.

[Hab97] P. Habermehl. "On the Complexity of the Linear-Time mu -calculus for Petri-Nets." In: *Proceedings of the 18th International Conference on Application and Theory of Petri Nets - ICATPN'97*. Vol. 1248. Lecture Notes in Computer Science. Springer, 1997, pp. 102–116.

[Hau90] D. Hauschildt. "Semilinearity of the reachability set is decidable for Petri nets." PhD thesis. University of Hamburg, Germany, 1990.

[Her91] M. Herlihy. "Wait-Free Synchronization." In: *ACM Transactions on Programming Languages and Systems* 13.1 (1991), 124–s149.

[HLM03] M. Herlihy, V. Luchangco, and M. Moir. "Obstruction-Free Synchronization: Double-Ended Queues as an Example." In: *Proceedings of ths 23rd International Conference on Distributed Computing Systems- ICDCS'03*. IEEE Computer Society, 2003, pp. 522–529.

[Hig52] G. Higman. "Ordering by Divisibility in Abstract Algebras." In: *Proceedings of the London Mathematical Society* s3-2.1 (1952), pp. 326–336.

[HP79] J. E. Hopcroft and J.-J. Pansiot. "On the Reachability Problem for 5-Dimensional Vector Addition Systems." In: *Theoretical Computer Science* 8 (1979), pp. 135–159.

[HS20] F. Horn and A. Sangnier. "Deciding the Existence of Cut-Off in Parameterized Rendez-Vous Networks." In: *Proceedings of the 31st International Conference on Concurrency Theory - CONCUR'20*. Vol. 171. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 46:1–46:16.

[Iba78] O. H. Ibarra. "Reversal-Bounded Multicounter Machines and Their Decision Problems." In: *J. ACM* 25.1 (1978), pp. 116–133.

[Iba+00] O. H. Ibarra, J. Su, Z. Dang, T. Bultan, and R. A. Kemmerer. "Conter Machines: Decidable Properties and Applications to Verification Problems." In: *Proceedings of Mathematical Foundations of Computer Science -MFCS'00*. Vol. 1893. Lecture Notes in Computer Science. Springer, 2000, pp. 426–435.

[Iba+02] O. H. Ibarra, J. Su, Z. Dang, T. Bultan, and R. A. Kemmerer. "Counter Machines and Verification Problems." In: *Theoretical Computer Science* 289.1 (2002), pp. 165–189.

[IS16] R. Iosif and A. Sangnier. "How Hard is It to Verify Flat Affine Counter Systems with the Finite Monoid Property?" In: *Proceedings of 14th International Symposium on the Automated Technology for Verification and Analysis - ATVA'16*. Vol. 9938. Lecture Notes in Computer Science. 2016, pp. 89–105.

[Jan90] P. Jancar. "Decidability of a Temporal Logic Problem for Petri Nets." In: *Theoretical Computer Science* 74.1 (1990), pp. 71–93.

[JS07] P. Jancar and Z. Sawa. "A note on emptiness for alternating finite automata with a one-letter alphabet." In: *Information Processing Letters* 104.5 (2007), pp. 164–167.

[Jur98] M. Jurdzinski. "Deciding the Winner in Parity Games is in UP \cap co-Up." In: *Information Processing Letters* 68.3 (1998), pp. 119–124.

[JL07] M. Jurdzinski and R. Lazic. "Alternation-free modal mu-calculus for data trees." In: *Proceedings of the 22nd IEEE Symposium on Logic in Computer Science - LICS'07*. IEEE Computer Society, 2007, pp. 131–140.

[KF94] M. Kaminski and N. Francez. "Finite-Memory Automata." In: *Theoretical Computer Science* 134.2 (1994), pp. 329–363.

[Kam68] H. W. Kamp. "Tense Logic and The Theory of Linear Order." PhD thesis. University of California, Los Angeles, 1968.

[KM69] R. M. Karp and R. E. Miller. "Parallel Program Schemata." In: *J. Comput. Syst. Sci.* 3.2 (1969), pp. 147–195.

[Kna28] B. Knaster. "Un théoréme sur les fonctions d'ensembles." In: *Annales de la Société Polonaise de Mathématique* 6 (1928), pp. 133–134.

[Kos82] S. R. Kosaraju. "Decidability of Reachability in Vector Addition Systems (Preliminary Version)." In: *Proceedings of the 14th Annual ACM Symposium on Theory of Computing - STOC'82*. ACM, 1982, pp. 267–281.

[KS88] S. R. Kosaraju and G. F. Sullivan. "Detecting Cycles in Dynamic Graphs in Polynomial Time (Preliminary Version)." In: *Proceedings of the 20th Annual ACM Symposium on Theory of Computing - STOC'88*. ACM, 1988, pp. 398–406.

[KS05] A. Kucera and J. Strejcek. "The stuttering principle revisited." In: *Acta Informatica* 41.7-8 (2005), pp. 415–434.

[KF11] L. Kuhtz and B. Finkbeiner. "Weak Kripke Structures and LTL." In: *Proceedings of the 22nd International Conference on Concurrency Theory - CONCUR'11*. Vol. 6901. Lecture Notes in Computer Science. Springer, 2011, pp. 419–433.

[LS10] S. Labbé and A. Sangnier. "Formal Verification of Industrial Software with Dynamic Memory Management." In: *Proceedings of the 16th IEEE Pacific Rim International Symposium on Dependable Computing - PRDC'10*. IEEE Computer Society, 2010, pp. 77–84.

[Lad75] R. E. Ladner. "The circuit value problem is log space complete for *P*." In: *SIGACT News* 7.1 (1975), pp. 18–20.

[LLT05] P. Lafourcade, D. Lugiez, and R. Treinen. "Intruder Deduction for *AC*-Like Equational Theories with Homomorphisms." In: *Proceedings of the 16th International Conference on Term Rewriting and Applications - RTA'05*. Vol. 3467. Lecture Notes in Computer Science. Springer, 2005, pp. 308–322.

[Lam92] J.-L. Lambert. "A Structure to Decide Reachability in Petri Nets." In: *Theoretical Computer Science* 99.1 (1992), pp. 79–104.

[LMS15] F. Laroussinie, N. Markey, and A. Sangnier. "ATLsc with partial observation." In: *Proceedings of the 6th International Symposium on Games, Automata, Logics and Formal Verification - GandALF'15*. Vol. 193. EPTCS. 2015, pp. 43–57.

[LNS04] F. Laroussinie, N., and P. Schnoebelen. "Model Checking Timed Automata with One or Two Clocks." In: *Proceedings of the 15th International Conference on Concurrency Theory - CONCUR'04*. Vol. 3170. Lecture Notes in Computer Science. Springer, 2004, pp. 387–401.

[Lec+16] A. Lechner, R. Mayr, J. Ouaknine, A. Pouly, and J. Worrell. "Model Checking Flat Freeze LTL on One-Counter Automata." In: *27th International Conference on Concurrency Theory, CONCUR 2016, Proceedings*. Vol. 59. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016, 29:1–29:14.

[Lec+18] A. Lechner, R. Mayr, J. Ouaknine, A. Pouly, and J. Worrell. "Model Checking Flat Freeze LTL on One-Counter Automata." In: *Log. Methods Comput. Sci.* 14.4 (2018).

[Ler11] J. Leroux. "Vector addition system reachability problem: a short self-contained proof." In: *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages - POPL'11*. ACM, 2011, pp. 307–316.

[LS15] J. Leroux and S. Schmitz. "Demystifying Reachability in Vector Addition Systems." In: *Proceedings of the 30th Annual ACM/IEEE Symposium on Logic in Computer Science- LICS'15*. IEEE Computer Society, 2015, pp. 56–67.

[LS04] J. Leroux and G. Sutre. "On Flatness for 2-Dimensional Vector Addition Systems with States." In: *Proceedings of the 15th International Conference on Concurrency Theory - CONCUR'04*. Vol. 3170. Lecture Notes in Computer Science. Springer, 2004, pp. 402–416.

[LS05] J. Leroux and G. Sutre. "Flat Counter Automata Almost Everywhere!" In: *Proceedings of the 3rd International Symposium on Automated Technology for Verification and Analysis - ATVA'05*. Vol. 3707. Lecture Notes in Computer Science. Springer, 2005, pp. 489–503.

[Lip76] R.J. Lipton. *The reachability problem requires exponential space*. Research report (Yale University. Department of Computer Science). Department of Computer Science, Yale University, 1976.

[MS03] N. Markey and P. Schnoebelen. "Model Checking a Path." In: *Proceedings of the 14th International Conference on Concurrency Theory - CONCUR'03*. Vol. 2761. Lecture Notes in Computer Science. Springer, 2003, pp. 248–262.

[May84] E. W. Mayr. "An Algorithm for the General Petri Net Reachability Problem." In: *SIAM J. Comput.* 13.3 (1984), pp. 441–460.

[May03] R. Mayr. "Undecidable problems in unreliable computations." In: *Theoretical Computer Science* 297.1-3 (2003), pp. 337–354.

[Min67] M. Minsky. *Computation, Finite and Infinite Machines*. Prentice Hall, 1967.

[MH84] S. Miyano and T. Hayashi. "Alternating Finite Automata on omega-Words." In: *Theor. Comput. Sci.* 32 (1984), pp. 321–330.

[NSV04] F. Neven, T. Schwentick, and V. Vianu. "Finite state machines for strings over infinite alphabets." In: *ACM Trans. Comput. Log.* 5.3 (2004), pp. 403–435.

[OW07] J. Ouaknine and J. Worrell. "On the decidability and complexity of Metric Temporal Logic over finite words." In: *Logical Methods in Computer Science* 3.1 (2007).

[Pap94] C. H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994. ISBN: 978-0-201-53082-7.

[Pet62] C.A. Petri. "Kommunikation mit Automaten." PhD thesis. Institut für instrumentelle Mathematik, 1962.

[PR90] A. Pnueli and R. Rosner. "Distributed Reactive Systems Are Hard to Synthesize." In: *Proceedings of the 31st Annual Symposium on Foundations of Computer Science - FOCS'90 - Volume II*. IEEE Computer Society, 1990, pp. 746–757.

[Pot91] L. Pottier. "Minimal Solutions of Linear Diophantine Systems: Bounds and Algorithms." In: *Proceedings of the 4th International Conference on Rewriting Techniques and Applications - RTA'91*. Vol. 488. Lecture Notes in Computer Science. Springer, 1991, pp. 162–173.

[Pre29] M. Presburger. "Uber die Vollstandigkeiteines gewissen Systems der Arithmetik ganzer Zahlen, in welchen die Addition als einzige Operation hervortritt." In: *Comptes-Rendus du ler Congres des Mathematiciens des Pays Slavs* (1929).

[Rac78] C. Rackoff. "The Covering and Boundedness Problems for Vector Addition Systems." In: *Theoretical Computer Science* 6 (1978), pp. 223–231.

[RSB05] J.-F. Raskin, M. Samuelides, and L. Van Begin. "Games for Counting Abstractions." In: *Electronic Notes in Theoretical Computer Science* 128.6 (2005), pp. 69–85.

[RS09] P.-A. Reynier and A. Sangnier. "Weak Time Petri Nets Strike Back!" In: *Proceedings of the 20th International Conference on Concurrency Theory - CONCUR'09*. Vol. 5710. Lecture Notes in Computer Science. Springer, 2009, pp. 557–571.

[RS04] N. Robertson and P. D. Seymour. "Graph Minors. XX. Wagner's conjecture." In: *J. Comb. Theory, Ser. B* 92.2 (2004), pp. 325–357.

[San08] A. Sangnier. "Vérification de systèmes avec compteurs et pointeurs." Thèse de doctorat. LSV, ENS Cachan, France, 2008.

[San+17] A. Sangnier, N. Sznajder, M. Potop-Butucaru, and S. Tixeuil. "Parameterized verification of algorithms for oblivious robots on a ring." In: *Proceedings of the 17th International Conference on Formal Methods in Computer Aided Design-FMCAD'17*. IEEE, 2017, pp. 212–219.

[San+20] A. Sangnier, N. Sznajder, M. Potop-Butucaru, and S. Tixeuil. "Parameterized verification of algorithms for oblivious robots on a ring." In: *Formal Methods in System Design* 56.1 (2020), pp. 55–89.

[Sch02] Ph. Schnoebelen. "Verifying lossy channel systems has nonprimitive recursive complexity." In: *Information Processing Letters* 83.5 (2002), pp. 251–261.

[Sch86] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley & Sons, Inc., 1986. ISBN: 0-471-90854-1.

[Seg06] L. Segoufin. "Automata and Logics for Words and Trees over an Infinite Alphabet." In: *Proceedings of the 20th International Workshop on Computer Science Logic - CSL'06*. Vol. 4207. Lecture Notes in Computer Science. Springer, 2006, pp. 41–57.

[Ser06] O. Serre. "Parity Games Played on Transition Graphs of One-Counter Processes." In: *Proceedings of the 9th International Conference on Foundations of Software Science and Computation Structures - FOSSACS'06*. Vol. 3921. Lecture Notes in Computer Science. Springer, 2006, pp. 337–351.

[SRS09] A. Singh, C. R. Ramakrishnan, and S. A. Smolka. "Query-Based Model Checking of Ad Hoc Network Protocols." In: *Proceeedings of the 20th International Conference on Concurrency Theory - CONCUR'09*. Vol. 5710. Lecture Notes in Computer Science. Springer, 2009, pp. 603–619.

[SC85] A. P. Sistla and E. M. Clarke. "The Complexity of Propositional Linear Temporal Logics." In: *Journal of the ACM* 32.3 (1985), pp. 733–749.

[Sto74] L. J. Stockmeyer. "The complexity of decision problems in automata and logic." PhD thesis. MIT, 1974.

[Tar55] A. Tarski. "A lattice-theoretical fixpoint theorem and its applications." In: *Pacific Journal of Mathematics* 5.2 (1955), pp. 285–309.

[VJ85] R. Valk and M. Jantzen. "The Residue of Vector Sets with Applications to Decidability Problems in Petri Nets." In: *Acta Informatica* 21 (1985), pp. 643–674.

[Var88] M. Y. Vardi. "A Temporal Fixpoint Calculus." In: *Proceedings of the 15th Annual ACM Symposium on Principles of Programming Languages - POPL'88*. ACM Press, 1988, pp. 250–259.

[VW86] M. Y. Vardi and P. Wolper. "An Automata-Theoretic Approach to Automatic Program Verification (Preliminary Report)." In: *Proceedings of the Symposium on Logic in Computer Science-LICS'86)*. IEEE Computer Society, 1986, pp. 332–344.

[WS09] J. Widder and U. Schmid. "The Theta-Model: achieving synchrony without clocks." In: *Distributed Computing* 22.1 (2009), pp. 29–47.

[Wol83] P. Wolper. "Temporal Logic Can Be More Expressive." In: *Information and Control* 56.1/2 (1983), pp. 72–99.

[Zie98] W. Zielonka. "Infinite Games on Finitely Coloured Graphs with Applications to Automata on Infinite Trees." In: *Theoretical Computer Science* 200.1-2 (1998), pp. 135–183.