AIX-MARSEILLE UNIVERSITÉ
ED 184 MATHEMATIQUES ET INFORMATIQUE
LABORATOIRE D'INFORMATIQUE ET SYSTEMES

Thèse présentée pour obtenir le grade universitaire de docteur

Discipline: Informatique

Damien Busatto-Gaston

Synthèse symbolique de contrôleurs pour systèmes temporisés: robustesse et optimalité

Symbolic controller synthesis for timed systems: robustness and optimality

Soutenue le 03/12/2019 devant le jury composé de:

Nathalie Bertrand	(CR) INRIA Rennes Bretagne-Atlantique	Examinatrice
Patricia Bouyer-Decitre	(DR) CNRS, LSV, ENS Paris-Saclay	Examinatrice
Krishnendu Chatterjee	(PR) IST Austria	Rapporteur
Benjamin Monmege	(MCF) LIS, CNRS, Aix Marseille Université	Directeur
Joël Ouaknine	(PR) MPI for Software Systems	Rapporteur
Laure Petrucci	(PR) LIPN, CNRS, Université Paris 13	Examinatrice
Pierre-Alain Reynier	(PR) LIS, CNRS, Aix Marseille Université	Directeur
Igor Walukiewicz	(DR) LaBRI, CNRS, Université de Bordeaux	Rapporteur

Remerciements

J'aimerais tout d'abord remercier Benjamin et Pierre-Alain pour le soutien constant et les conseils précis qu'ils m'ont fournis au cours de cette thèse. Ces trois années enrichissantes furent un plaisir par leur encadrement et leur bienveillance.

Je voudrais également remercier Krishnendu Chatterjee, Joël Ouaknine et Igor Walukiewicz pour avoir été rapporteurs de ce document, ainsi que Nathalie Bertrand, Laure Petrucci et Patricia Bouyer-Decitre pour s'être intéressées à mes travaux en tant que membres du jury. Je remercie en particulier cette dernière pour m'avoir fait découvrir ce sujet, m'avoir conseillé et m'avoir orienté vers Marseille lorsque j'étais en master.

Si le nom, les locaux et la direction du LIS ont changé au cours de ces dernières années, il est resté un laboratoire particulièrement chaleureux et agréable, dont j'aimerais remercier les membres permanents pour leur acceuil, et les non-permanents pour leur complicité et leur entrain. Merci à mes co-bureaux Florian, Didier, Eloi d'avoir supporté ma tendance récurrente à faire les cents pas. Je remercie particulièrement Sébastien R. pour m'avoir assuré sans failles, ainsi que Théodore, Amélia, Manon P., Léo E., José Luis, Thibault, Manon S., Cindy, Franck, Pacôme, Jeremy, Cedric, Sébastien D. et les autres pour nos échanges scientifiques ou ludiques réguliers. Je remercie aussi mes amis, et en particulier Léo T. et Antoine pour leur coopération.

Je remercie enfin ma famille pour leurs encouragements constants, leurs efforts pour s'intéresser à mes travaux et leur bonne humeur.

Résumé

Le domaine de la synthèse réactive a pour objectif d'obtenir un système correct par construction à partir d'une spécification logique. Une approche classique consiste à se ramener à un jeu à somme nulle, où deux joueurs interagissent tour-à-tour dans un système de transitions, et à se demander si le joueur "contrôleur" peut garantir que son objectif sera rempli, et ce indépendamment des décisions du joueur "environnement". Nous étudions des spécifications temps-réel, modélisées par un automate temporisé équipé d'un objectif d'accessibilité ou de Büchi, et présentons des méthodes symboliques pour synthétiser des stratégies du contrôleur. Nos contributions concernent deux problématiques distinctes : on peut souhaiter que le contrôleur obtienne une stratégie robuste aux perturbations, ou bien le faire jouer de manière optimale dans un jeu pondéré. Dans le contexte de la robustesse, le contrôleur a pour objectif de suivre un lasso acceptant de l'automate. De plus, ses choix de délais successifs doivent résister à d'éventuelles perturbations choisies par l'environnement. Ce problème est connu comme étant PSPACE-complet, mais les techniques existantes opèrent sur l'abstraction des régions. Nous proposons une solution moins sensible à une explosion de l'espace d'états, en faisant exclusivement usage de zones. Dans le contexte quantitatif, nous étudions des jeux sur automates temporisés équipés de poids : le contrôleur souhaite minimiser le poids accumulé en atteignant un état cible, alors que l'environnement vise l'objectif opposé. Dans une perspective de synthèse réactive, cette extension pondérée des jeux temporisés permet de mesurer le degré de qualité du contrôleur. Les jeux temporisés pondérés sont rapidement indécidables, même lorsque les poids sont tous positifs. Des résultats de décidabilité existent pour une classe à poids positifs, définie par une restriction sémantique sur le poids des cycles. Nous introduisons la classe des jeux temporisés pondérés divergents comme une généralisation de cette restriction sémantique autorisant les poids négatifs, ce qui permet de représenter par exemple de l'énergie ou de l'argent. Nous présentons une méthode pour calculer la valeur optimale de ces jeux. Les jeux divergents forment donc la première classe décidable de jeux temporisés pondérés avec des poids négatifs et un nombre arbitraire d'horloges. Nous étudions enfin une classe plus générale introduite par Bouyer, Jaziri et Markey en 2015, qui reste analysable lorsque les poids sont positifs. Bien que cette classe soit indécidable, les auteurs montrent que l'on peut approximer la valeur du jeu en utilisant des régions de granularité moindre. Nous étendons cette classe pour autoriser des poids négatifs, et montrons que la valeur y reste approximable. De plus, nous expliquons qu'un algorithme symbolique suivant le paradigme de *value iteration* peut être utilisé sur cette classe en tant que schéma d'approximation.

Mots clés : Automates temporisés, Synthèse, Robustesse, Jeux pondérés

Abstract

The field of reactive synthesis studies ways to obtain, starting from a specification, a system that is correct by construction. A classical approach models this setting as a zero-sum game played by two players on a transition system, and asks whether player controller can ensure an objective against any competing player environment. We focus on real-time specifications, modelled as timed automata with reachability or Büchi acceptance conditions, and present symbolic ways to synthesise strategies for the controller. We consider two problems, either restricting controller to robust strategies or aiming for optimal strategies in a weighted game setting. In the robustness setting, the goal of the controller is to play according to an accepting lasso of the automaton, while resisting to timing perturbations chosen by a competing environment. The problem was previously shown to be PSPACE-complete using regions-based techniques, but we provide a first tool solving the problem using zones only, thus more resilient to state-space explosion issues. In the quantitative setting, we study games played on a timed automaton equipped with weights: controller wants to minimise the accumulated weight while reaching a target, while the environment has an opposite objective. In a reactive synthesis perspective, this quantitative extension of timed games allows one to measure the quality of controllers. Weighted timed games are notoriously difficult and quickly undecidable, even when restricted to non-negative weights. Decidability results exist for a subclass with non-negative weights defined by a semantical restriction on the weights of cycles. We introduce the class of divergent weighted timed games as a generalisation of this semantical restriction to arbitrary weights, allowing one to model energy for instance. We show how to compute their optimal value, yielding the first decidable class of weighted timed games with negative weights and an arbitrary number of clocks. Then, we focus on a larger class, known to be analysable with non-negative weights, that has been introduced by Bouyer, Jaziri and Markey in 2015. Though the value problem is undecidable, the authors show how to approximate the value by considering regions with a refined granularity. We extend this class to incorporate negative weights, and prove that the value can still be approximated. In addition, we show that a symbolic algorithm, relying on the paradigm of value iteration, can be used as an approximation schema on this class.

Keywords: Timed automata, Synthesis, Robustness, Weighted games

Contents

Introduction

The widespread use of digital technology can be felt in most aspects of modern life. Digital systems are designed to assist with—and sometimes perform autonomously—important tasks, from long-distance communications to the management of our electrical or financial infrastructures. When faced with a known situation, we trust these systems to act accordingly, with near-instantaneous reaction times. This holds particularly for *embedded systems*, *i.e.* systems designed to perform a fixed task, as part of a broader structure. They can be found in most electronic devices, from phones and home appliances to traffic lights, avionics and elevators. This has prompted a large research effort in computer science, where a series of work has been devoted to the creation, analysis and testing of embedded systems, as they are sometimes entrusted with our safety.

This is a challenging task, as the correctness or reliability of a system can depend on fine interactions with its surroundings or with other systems, and it can be hard, for the human designer, to anticipate on all possible behaviours.

Computer-aided verification

One approach lies in the field of *formal methods*, where one studies a mathematical idealisation of the system, called a *model*, to check that it satisfies some desired properties, such as eventually performing a particular task, or avoiding errors. The objective is to develop automated methods to analyse and verify systems. For example, the model-checking problem asks if a given model satisfies its *specification*, described by a logical formulæ. This problem has been extensively studied in various settings, with practical tools being used for the verification of hardware and software in industry (see *e.g.* [CHVB18]).

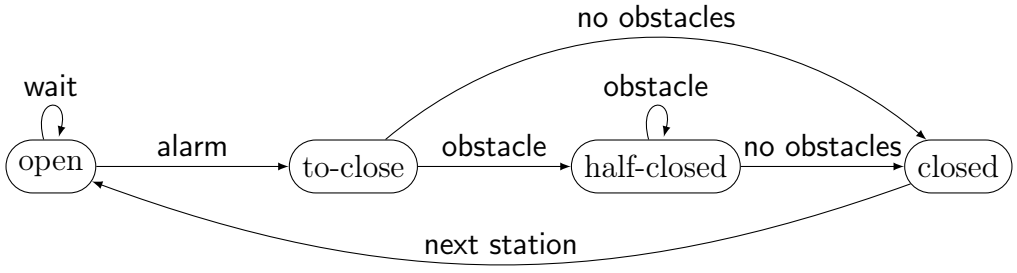


Figure 1.: A transition system modelling train doors.

Example 1. As an example, we will use an embedded system that controls a pair of doors in a subway train, represented in Figure 1. Whenever the train arrives at a new station, the doors open. The system waits until an alarm signals departure for the next station, then checks for any obstacles to their closing. If such an obstacle is present, the doors close partially, and wait until the obstacle is removed before closing. Desired property for the system might be "the train must always travel with closed doors", or "doors always eventually close". The former is always guaranteed by this model, but not the latter, that requires additional assumptions (we do not wait forever for the departure signal, and obstacles are always eventually removed).

An ambitious variant is that of *controller synthesis*, where one starts from the specification alone. The goal is to build a model that will, by construction, be correct. A classical approach expresses this problem from the viewpoint of game theory: One encodes all possible behaviours in a *transition system*, where two players are opposed. Starting from some initial state for the system, the player named *controller* makes decisions that determine how the system dynamically evolves. Thus, a sequence of successive states is obtained, and we want this sequence to satisfy the specification. Examples include asking that a target "good" state is eventually reached, or conversely that an error "bad" state is never reached. However, the controller may sometimes not get to choose the next state, in which case we say that the decision is made by the second player, called the *environment*. The notion of environment captures everything that cannot be anticipated, from user inputs to complex interactions with the real world or foreign systems. Overall, the way the system evolves is derived from the choices of both players in a turn-based fashion, and this interaction forms a game, where controller wins if its objective is met. Our goal is thus to automatically construct a *strategy* for the controller, that is a recipe dictating how to play, so that controller wins no matter how the environment plays. Such strategy finally describes a system that is necessarily correct, since the specification must hold in all possible scenarios.

Example 2. Going back to the example of Figure 1, one could consider that whether an obstacle is present or not is not under our control. As such, player controller can choose whether to wait, ring the alarm, or go to the next station when in the appropriate states, but player environment is the one that decides if there is an obstacle or not.

Timed systems

We focus on a class of programs sensitive to real-time, where keeping track of how much time elapses between the decisions taken by the program is required to differentiate the good and bad behaviours. This is a common requirement for embedded systems, as they interact with the real world. The design of such programs is a notoriously difficult problem, because they must take care of delicate timing issues, and are difficult to debug a posteriori. In order to ease the design of real-time software, it appears important to automatise the process by using formal methods. The situation may be modelled into a *timed automaton* [AD94], namely a transition system equipped with

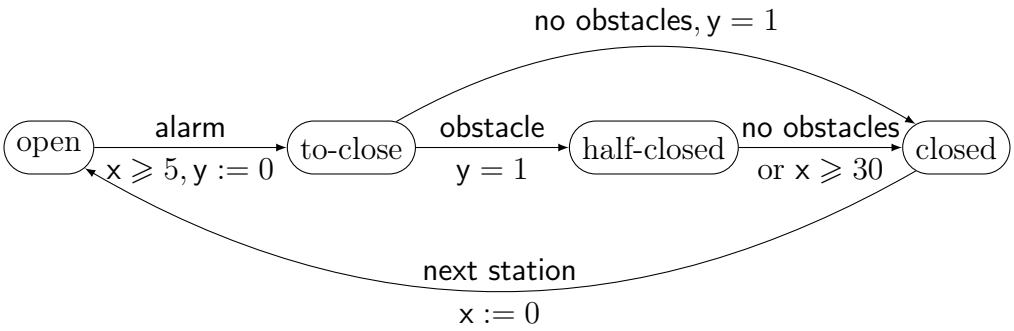


Figure 2.: A timed automaton modelling train doors.

real-valued variables, called clocks, evolving with a uniform rate. Transitions are equipped with timing constraints expressed over the clocks, and may only be taken when these constraints are met.

Example 3. We enrich the train door example from Figure 1, in order to obtain the timed automaton depicted on Figure 2. Clocks x, y are variables that continuously increase, at a rate of one unit per second. Whenever the train arrives at a new station, clock x is *reset* (*i.e.* set to 0). We must stay in the open state at least five seconds before signalling departure, at which point clock y is reset. Exactly one second later, the system closes the doors if possible, and otherwise goes to the half-closed state. We then wait until either the obstacle is removed or the time since arrival at the current station (recorded by x) is at least thirty seconds, in which case we forcefully close the doors and leave.[1] Note that we removed the waiting loops in the open and half-closed states, as they are no longer needed: in a timed automaton, one can always stay in the current state and let time elapse.

In order to verify the real-time system, one determines whether there exists an accepting execution in the timed automaton. A simple, yet realistic specification asks that a target state is reached at some point. We are also interested in Büchi acceptance conditions, where a target should be reached infinitely often along the execution, modelling cases where we do not want the system to get stuck in a bad situation. It has been proven in [AD94] that the reachability and Büchi problems on timed automata are both PSPACE-complete, by partitioning the state space into *regions*. While optimal from a theoretical complexity point of view, practical tools tend to favour efficient *symbolic* algorithms for solving these problems, that use *zones* instead of regions, as they allow an on-demand partitioning of the state space. This leads to much better performances, as witnessed by successful model-checking tools like Uppaal [LPY97], Kronos [BDM+98], or TChecker [HPT19, HSW10].

In this thesis, we study two controller synthesis problems on timed automata, either restricting controller to *robust* strategies or aiming for optimal strategies in a *weighted game* setting. The document is split into three parts:

[1] Do not imitate in real life.

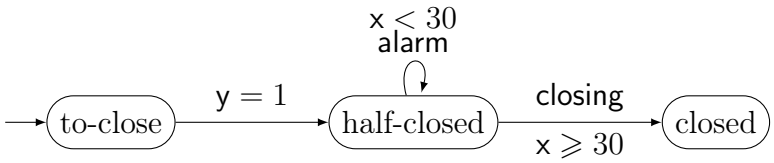


Figure 3.: A timed automaton with Zeno behaviours.

- In Part I, we recall formalisms and known techniques for the study of finite or timed transition systems;
- In Part II, we focus on making choices that are robust with regard to small timing perturbations;
- In Part III, we present results on weighted timed games, where one aims for optimal choices in a quantitative setting.

Robustness

As we have seen, timed automata [AD94] provide an automata-theoretic framework to design, model and verify real-time systems. However, the semantics of timed automata is a mathematical idealisation: it assumes that clocks have infinite precision and that actions are instantaneous. Proving that a timed automaton satisfies a property may not ensure that a real implementation of it also does. This *robustness* issue is a challenging problem for embedded systems [HS06], and alternative semantics have been proposed, so as to ensure that the verified (or synthesised) behaviour remains correct in presence of small timing perturbations.

Example 4. As an example, consider Figure 3, that models part of a variant of Figure 2: We assume that there is an obstacle that is not removed, and we add an action that lets us repeat the alarm signal until the thirty seconds mark has been reached. We argue that some infinite behaviours of this system are not realistic, as it is possible to ring the alarm infinitely many times with a so-called Zeno behaviour. Indeed, one could let one second elapse, and use the alarm transition once. Then, we let half a second elapse, and use it again. By using increasingly smaller delays ($1/2^i$ after i steps), the alarm can be rung arbitrarily many times, while keeping the total time elapsed under two seconds.

Another issue lies in the steps that require infinite precision, like the transition from the to-close to the half-closed state. Indeed, it requires clock y to be valued at exactly 1. This might prove hard to ensure with a real system, and it would be preferable in this case to leave some margin $\eta > 0$ around 1, such that one asks for $\mathsf{y} \in [1-\eta, 1+\eta]$ instead.

We are interested in a fundamental model-checking problem in timed automata equipped with a Büchi condition: it consists in determining whether there exists an

accepting infinite execution. This problem has been studied numerously in the exact setting, where symbolic methods have been developed [TYB05, Tri09, Li09, HS10, HSW12, LOD+13, HSTW16]. In the context of robustness, the execution should be tolerant to small perturbations of the delays. This discards executions suffering from weaknesses such as Zeno behaviours, or even non-Zeno behaviours requiring infinite precision, as exhibited in [CHR02].

More formally, the semantics we consider is defined in Chapter 3 as a game that depends on some parameter δ representing an upper bound on the amplitude of the perturbation [CHP11]. In this game, the controller plays against an antagonistic environment that can perturb each delay using a value chosen in the interval $[-\delta, \delta]$. The case of a fixed value of δ has been shown to be decidable in [CHP11], and also for a related model in [LLTW14]. However, these algorithms are based on regions, and as the value of δ may be much smaller than the constants appearing in the guards of the automaton, do not yield practical algorithms. Moreover, the maximal perturbation is not necessarily known in advance, and could be considered as part of the design process.

The problem we are interested in is *qualitative*: we want to determine whether *there exists* a positive value of δ such that the controller wins the game. It has been proven in [SBMR13] that this problem is in PSPACE (and even PSPACE-complete), thus no harder than in the exact setting with no perturbation allowed. However, the algorithm, recalled in Chapter 4, heavily relies on regions, and more precisely on an abstraction that refines the one of regions, namely folded orbit graphs. Hence, it is not amenable to implementation, and has indeed never been implemented. Our main contribution in Part II is to provide, in Chapter 5, an efficient symbolic algorithm for solving this problem.

Our algorithm can be understood as an adaptation to the robustness setting of the standard algorithm for Büchi acceptance in timed automata [LOD+13]. This algorithm looks for an accepting lasso using a nested breadth-first search. A major difficulty consists in checking whether a lasso can be robustly iterated, *i.e.* whether there exists $\delta > 0$ such that the controller can follow the cycle for an infinite amount of steps while being tolerant to perturbations of amplitude at most δ.

Our approach relies on several new ingredients:

- We provide a polynomial time procedure to decide, given a lasso, whether it can be robustly iterated. This symbolic algorithm relies on a computation of the greatest fixpoint of the operator describing the set of controllable predecessors of a path. In order to provide an argument of termination for this computation, we resort to a new notion of branching constraint graphs, extending the approach used in [JR11, Tra16] and based on constraint graphs (introduced in [CLJ99]) to check iterability of a cycle, without robustness requirements.

- We provide a termination criterion for the analysis of lassos. Focusing on zones is not complete: it can be the case that two cycles lead to the same zone, but one is robustly iterable while the other one is not. Robust iterability crucially depends on the real-time dynamics of the cycle and we prove that it actually only depends on

the reachability relation of the path. We provide a polynomial-time algorithm for checking inclusion between reachability relations of paths in timed automata based on constraint graphs.

- It is worth noticing that all our procedures can be implemented using difference bound matrices, a very efficient data structure used for timed systems. These developments have been integrated in a tool, and we present a case study of a train regulation network illustrating its performances.

We finally study in Chapter 6 a *quantitative* variant, where one computes the greatest value of δ such that the controller wins the game. We obtain a new decidability result for this problem, by showing that when considering a lasso, not only can we decide robust iterability, but we can even compute the largest perturbation under which it is controllable.

Weighted timed games

Solving the robustness game for a fixed value of δ can be seen as an instance of a more general problem, where both players alternatively choose transitions and delays. This is a natural extension of the controller synthesis problem to the real-time setting called a *timed game*, where a controller and an antagonistic environment play on a timed automaton instead of a finite transition system. Strategies of players become recipes dictating how to play on this arena (timing delays and transitions to follow). In this more ambitious setting, we will focus on reachability objectives, and we are thus looking for a strategy of the controller so that the target is reached no matter how the environment plays. Reachability timed games are decidable [AM99], and EXPTIME-complete [JT07].

If the controller has a winning strategy in a given reachability timed game, several such winning strategies could exist. Weighted extensions of these games have been considered in order to measure the quality of the winning strategy for the controller [BCFL04]. This means that the game, defined in Chapter 8, now takes place over a *weighted (or priced) timed automaton* [BFH+01, ALTP04], where edges are equipped with weights, and locations with rates of weights (the cost is then proportional to the time spent in this location, with the rate as proportional coefficient).

Example 5. As a motivating example for studying weighted games, we present a ride-sharing scenario. As a driver, we wish to travel from point A to point B, and must choose between several options, as displayed in Figure 4. We can use a direct road, and reach B in two to three hours, or an highway that lets us reach our destination in one hour. Alternatively, we can make a detour: another traveller is waiting at point C, and wishes to reach point D. For this portion too, a faster highway is available.

While all four possible paths satisfy the objective "reaching B", we want to select the one that lets us spend as little money as possible for the trip. The cost of each path depends on several factors. There are fixed entry fees (of 1€) for the highways, and we need to keep track of fuel consumption, as the rate at which fuel is used differs in roads

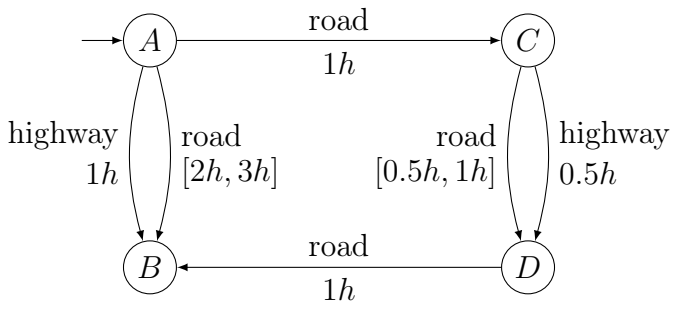


Figure 4.: A ride-sharing decision diagram.

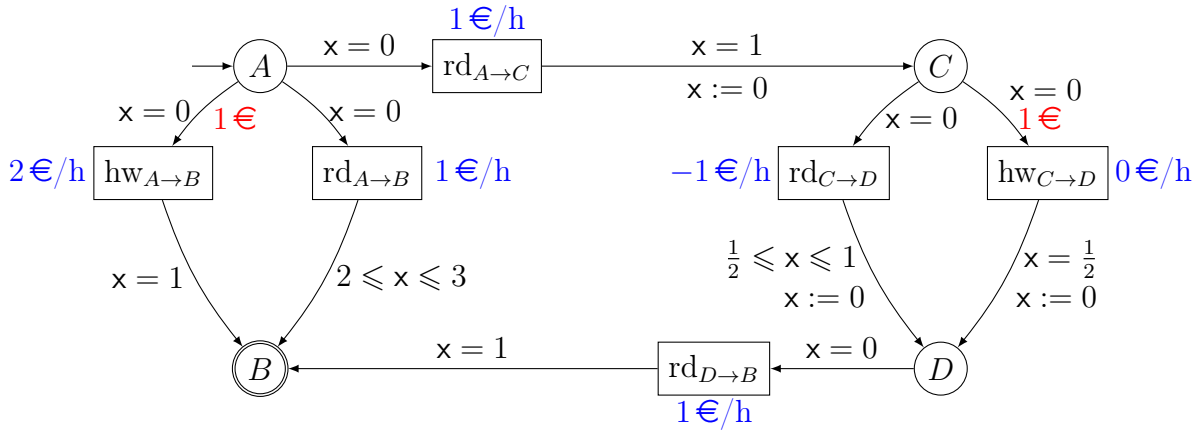


Figure 5.: A weighted timed game modelling Figure 4. The cost of waiting in a state is displayed in blue, the cost of taking a transition is in red. States and transitions without costs have a weight of 0 €

and highways. Thus, we say that roads cost 1 € per hour, while highways cost 2 € per hour. Moreover, if we share the portion from C to D, the other traveller will pay for his trip (at a rate of 2 €/h), and that can lower our costs. A shared road therefore costs us -1 € per hour (negative rate means we are making a profit), while a shared highway costs 0 €/h.

The situation can be modelled as a weighted timed game, displayed in Figure 5. The controller chooses delays and transitions in circle states, while the environment controls the square ones. For example, if we choose to use the direct road from A to B, we go (immediately) to state $\mathrm{rd}_{A\to B}$, and stay there until going to state B. This requires letting between two and three hours elapse in $\mathrm{rd}_{A\to B}$, with a cost of 1 €/h. The delay is chosen by the environment, as it depends on external influences like traffic density.

In this example, the optimal strategy is to share the road from C to D. This lets us ensure a total weight of at most 1.5 €: going from A to C costs 1 € in the worst case; going from D to B similarly costs at most 1 €; and sharing the trip from C to D is guaranteed to bring us at least 0.5 €.

While solving weighted timed automata has been shown to be PSPACE-complete [BBBR07] (*i.e.* the same complexity as the non-weighted version), weighted timed games are known to be undecidable [BBR05]. This has led to many restrictions in order to regain decidability, the first and most interesting one being the class of strictly non-Zeno cost with only non-negative weights (in edges and locations) [BCFL04]: this hypothesis states that every execution of the timed automaton that follows a cycle of the region abstraction has a weight far from 0 (in interval $[1, +\infty)$, for instance).

Less is known for weighted timed games in the presence of negative weights in edges and locations. In particular, no results exist so far for a class that does not restrict the number of clocks of the timed automaton to 1. However, negative weights are particularly interesting from a modelling perspective, for instance in case weights represent the consumption level of a resource (money, energy...) with the possibility to spend and gain some resource. In Chapter 9, we introduce a generalisation of the strictly non-Zeno cost hypothesis in the presence of negative weights, that we call *divergence*. We show the decidability of the class of divergent weighted timed games for the optimal synthesis problem in Chapter 10:

- We describe a procedure to solve weighted timed games for a bounded horizon, *i.e.* when controller has a fixed number of steps to reach his targets. It follows closely the framework of [ABM04], but is more symbolic and allows for negative weights.
- We show that optimal strategies in divergent weighted timed games can be restricted to a bounded horizon, that matches the one obtained in the non-negative case from the study of [BCFL04].

The techniques providing these decidability results cannot be extended if the conditions are slightly relaxed. For instance, if we add the possibility for an execution of the timed automaton following a cycle of the region automaton to have weight *exactly 0*, the decision problem is known to be undecidable [BJM15], even with non-negative weights only. For this extension, in the presence of non-negative weights only, it has been proposed an approximation schema to compute arbitrarily close estimates of the optimal weight that the controller can guarantee [BJM15]. To this end, the authors consider regions with a refined granularity so as to control the precision of the approximation.

Our contribution on the approximation front is presented in Chapter 11, and is two-fold:

- We extend the class considered in [BJM15] to the presence of negative weights, and provide an approximation schema for the resulting class of *almost-divergent games*;
- We show that the approximation can be obtained using a symbolic computation, that avoids an a priori refinement of regions.

Moreover, the classes of weighted timed games that we study induce interesting classes of finite weighted game when there are no clocks, that can be solved with a lower complexity than arbitrary weighted games. We present those results in Chapter 7.

Part I.

Controller synthesis and timed systems

1. Finite systems

Let us now formally introduce transition systems and their quantitative or game-theoretical extensions. In this chapter we study finite systems only, but terminology and notations are defined over infinite ones with further chapters in mind.

1.1. Transition systems

Definition 1.1. Let Σ be a set of elements called labels. A *transition system* labelled over Σ is a pair $\langle S, T\rangle$, with S a set of states and $T \subseteq S \times \Sigma \times S$ a set of transitions, such that $(s, a, s') \in T$ is denoted $s \xrightarrow{a} s'$.

A transition system is finite if it has finitely many states and transitions. A directed graph labelled over Σ is a finite transition system $\langle S, T\rangle$ (labelled over Σ), such that T contains at most one transition from s to s' for every pair of state (s, s'). A relation R over a domain Q is a set of pairs in $Q \times Q$, and we sometimes write $a \,\mathrm{R}\, b$ to denote that the pair (a, b) belongs to R. A relation is complete if it equals $Q \times Q$. A graph $\langle S, T\rangle$ induces a relation over the finite domain S as $\{(s, s') \mid \exists a \in \Sigma, s \xrightarrow{a} s'\}$: the set of states (s, s') linked by a transition. It is complete if this relation is complete.

For $k \geqslant 1$, a finite path of length k is a finite sequence of transitions $(s_i, a_i, s'_i)_{1 \leqslant i \leqslant k}$ such that for all $i \in [1, k-1]$, $s'_i = s_{i+1}$. Such a path ρ will be denoted $s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \dots s_k \xrightarrow{a_k} s'_k$, and is said to be a path from state $\mathsf{first}(\rho) = s_1$ to state $\mathsf{last}(\rho) = s'_k$ of length $|\rho| = k$. The concatenation of two finite paths ρ_1 and ρ_2, such that $\mathsf{last}(\rho_1) = \mathsf{first}(\rho_2)$, is denoted by $\rho_1\rho_2$. Transitions can be seen as paths of length one and states as paths of length zero, and we extend the first and last operators in those cases. A cycle is a finite path ρ, of length at least 1, such that $\mathsf{first}(\rho) = \mathsf{last}(\rho)$.

We similarly define an infinite path $\rho \in T^{\mathbb{N}}$ as an infinite sequence of transitions $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \dots$, with $\mathsf{first}(\rho) = s_0$. A state s is called a deadlock state if there are no transitions $t \in T$ with $\mathsf{first}(t) = s$. A path ρ is maximal if it is infinite or if it is finite and ends in a deadlock state. Conversely, a non-maximal path is a finite path that can be extended.

Example 1.1. Figure 1.1 represents a transition system labelled over $\{\mathsf{req}, \mathsf{rec}, \mathsf{ok}, \mathsf{pro}, \mathsf{slp}, \mathsf{to}\}$, that models a client interacting with a server. From an initial state s_0, the client sends a *request* for data by taking the transition labelled by req. The server should answer by sending a finite sequence of data, activating the *receive* transition rec multiple times. The communication should end with a *confirmation* as the ok transition, allowing the client to *process* the data with pro and return to the initial state. The client can *sleep* in the initial state with slp, and if the communication fails the client will *time-out* with transition to.

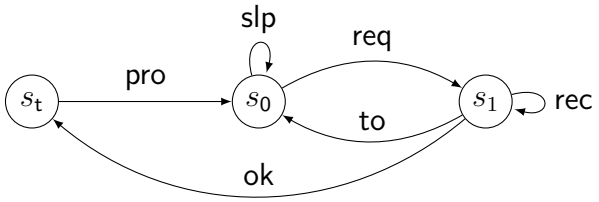


Figure 1.1.: A transition system modelling a client that requests a sequence of data to a server, receives it and processes it.

In this analogy, the finite path $s_0 \xrightarrow{\mathsf{req}} s_1 \xrightarrow{\mathsf{rec}} s_1 \xrightarrow{\mathsf{ok}} s_t \xrightarrow{\mathsf{pro}} s_0 \xrightarrow{\mathsf{req}} s_1 \xrightarrow{\mathsf{to}} s_0$ represents an execution of the client where a successful interaction is followed by a time-out. There are no deadlock states in this transition system, and therefore every maximal path is infinite and every finite-path is non-maximal.

A finite (resp. infinite) path $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \dots$ naturally induces a finite (resp. infinite) word $a_0 a_1 \dots$ over labels. A set of infinite words $\mathcal{L}$ is called an ω-language. Given a transition system and an initial state s_0, the ω-language $\mathcal{L}_{s_0}$ is defined as the set of infinite words over labels induced by the infinite paths that start from s_0. If the transition system is finite, it can be seen as a non-deterministic Büchi automata where every state is accepting, and therefore its language is an ω-regular language, a notion that generalises the notion of regular languages to infinite words.

Definition 1.2. Given a transition system $\langle S, T \rangle$, an objective $\mathcal{L}_t$ (as an ω-language over labels) and an initial state s_0, the *emptiness problem* asks if $\mathcal{L}_{s_0} \cap \mathcal{L}_t \neq \emptyset$, *i.e.* is there an infinite path starting from s_0 that induces a word in $\mathcal{L}_t$.

Definition 1.3. Given a transition system $\langle S, T \rangle$, an objective $\mathcal{L}_t$ (as an ω-language over labels) and an initial state s_0, the *model-checking problem* asks if $\mathcal{L}_{s_0} \subseteq \mathcal{L}_t$, *i.e.* does every infinite path starting from s_0 induce a word in $\mathcal{L}_t$.

The objective language encodes a specification, and can be expressed as an ω-regular expression or a formula in linear temporal logic (LTL) for example. We will be focusing on simple objectives, like the reachability of a target or a Büchi condition.

Definition 1.4. Given a transition system $\langle S, T \rangle$, an initial state $s_0 \in S$ and a set of target states $S_t \subseteq S$, the *emptiness problem with a reachability condition* asks if there exists a finite path ρ such that $\mathsf{first}(\rho) = s_0$ and $\mathsf{last}(\rho) \in S_t$. The *emptiness problem with a Büchi condition* asks if there exists an infinite path $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \dots$ starting from s_0 that reaches S_t infinitely often, *i.e.* such that there exists infinitely many $i \in \mathbb{N}$ with $s_i \in S_t$.

One can observe that the emptiness problem with a reachability (resp. Büchi) condition is a particular case of the emptiness problem, as we could extend if needed the label of every transition t with the states $\mathsf{first}(t)$ and $\mathsf{last}(t)$ and thus express these conditions on words over labels. We call the associated ω-language a reachability (resp. Büchi) objective.

Example 1.2. Consider the transition system in Figure 1.1, the initial state s_0 and the target states $S_t = \{s_t\}$. The emptiness problem with reachability condition S_t is satisfied, as s_t is reachable from s_0 by following req and ok. Since s_t can return to s_0 with pro, the emptiness problem with Büchi condition S_t is also satisfied. The corresponding reachability (resp. Büchi) objective is the ω-language of all infinite words over $\{\mathsf{req}, \mathsf{rec}, \mathsf{ok}, \mathsf{pro}, \mathsf{to}\}$ that contain at least one (resp. infinitely many) ok. The model-checking problem is not satisfied with these objectives, as it is possible to loop between s_0 and s_1 infinitely many times for example, never reaching s_t.

These problems have been well studied in a variety of settings, and we now give a few results that apply to finite systems only. The emptiness problem with a reachability condition can be solved in linear time using forward exploration techniques, like the classical breadth-first search algorithm. Such techniques can indeed compute the set of states reachable from s_0, which is enough for reachability conditions. For Büchi conditions, one can then launch a second exploration from every reachable state s_t in S_t and search for a loop around s_t. A linear time complexity can be obtained by computing all strongly connected components. Model checking is polynomial for reachability and Büchi objectives, as it can be solved by attractor computations (see Section 1.3.1).

1.2. Weighted transition systems

All of the problems defined so far have been qualitative in nature. In order to model quantitative notions like the cost of going from a source to a destination, we need to define concepts such as the weight of a path, or the weight of a set of paths. This can be done by considering a setting where labels are real numbers, the weight of a path is the sum of its labels and the weight of a set of paths is the minimum of the weight of each path. This is the setting that we will consider in our study of timed systems, and our problems can be seen as extensions of the classical shortest path problem. However, some developments (Difference Bound Matrices with non-standard entries) will require the same notions over more exotic labels, so we introduce the more general context of [BT10], where labels are only required to form a set with some algebraic properties.

1.2.1. Semirings, closure operation

A binary operation $\oplus$ over a domain Q is a mapping from $Q \times Q$ to Q, and we sometimes write $a \oplus b$ to denote the element $\oplus(a, b)$.

Definition 1.5. A *semiring* $(Q, \oplus, \otimes, \mathbf{0}, \mathbf{1})$ is a set Q equipped with two binary operations $\oplus$ and $\otimes$ over Q of respective neutral element $\mathbf{0}$ and $\mathbf{1}$, such that for all $a, b, c \in Q$:

- ($\oplus$ is associative) $(a \oplus b) \oplus c = a \oplus (b \oplus c)$
- ($\mathbf{0}$ is neutral for $\oplus$) $\mathbf{0} \oplus a = a \oplus \mathbf{0} = a$
- ($\oplus$ is commutative) $a \oplus b = b \oplus a$

- ($\otimes$ is associative) $(a \otimes b) \otimes c = a \otimes (b \otimes c)$
- ($\mathbf{1}$ is neutral for $\otimes$) $\mathbf{1} \otimes a = a \otimes \mathbf{1} = a$
- ($\otimes$ distributes over $\oplus$, left) $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$
- ($\otimes$ distributes over $\oplus$, right) $(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$
- ($\mathbf{0}$ is absorbing for $\otimes$) $\mathbf{0} \otimes a = a \otimes \mathbf{0} = \mathbf{0}$

Examples of semirings include the natural semiring $(\mathbb{N}, +, \times, 0, 1)$, equipped with standard addition and multiplication over integers, the tropical semiring $(\mathbb{N} \cup \{+\infty\}, \min, +, +\infty, 0)$, or the boolean semiring $(\{0, 1\}, \vee, \wedge, 0, 1)$. As the neutral elements $\mathbf{0}$ and $\mathbf{1}$ are uniquely determined by $\oplus$ and $\otimes$ [1], we may omit them and refer to the semiring as $(Q, \oplus, \otimes)$, or simply Q if the operations are clear from context. If $\mathbf{0} = \mathbf{1}$ then necessarily $Q = \{\mathbf{0}\}$, and Q is called the trivial semiring. The classical notion of ring additionally requires that every element of Q has an inverse by $\oplus$ in Q, *i.e.* $\forall a \in Q, \exists (-a) \in Q$ such that $a \oplus (-a) = \mathbf{0}$. Semirings strictly generalise rings, as for examples the boolean semiring $(\{0, 1\}, \vee, \wedge)$ is not a ring.

Definition 1.6. A relation $\sqsubseteq \in Q \times Q$ is a (partial) order if for all $a, b, c \in Q$

- ($\sqsubseteq$ is reflexive) $a \sqsubseteq a$
- ($\sqsubseteq$ is transitive) $a \sqsubseteq b \wedge b \sqsubseteq c \Rightarrow a \sqsubseteq c$
- ($\sqsubseteq$ is antisymmetric) $a \sqsubseteq b \wedge b \sqsubseteq a \Rightarrow a = b$

An *ordered semiring* is a semiring $(Q, \oplus, \otimes)$ such that the relation $\sqsubseteq$ defined as $\{(a, b) \mid \exists c \in Q, a \oplus c = b\}$ is an order on Q. The relation $\sqsubseteq$ is always reflexive and transitive by the properties of $\oplus$, but it is not antisymetric on every semiring. In fact, a non-trivial ring cannot be an ordered semiring. For example, $(\mathbb{R}_{\geqslant 0}, +, \times)$ is an ordered semiring with $\sqsubseteq$ equal to the standard $\leqslant$ order on $\mathbb{R}_{\geqslant 0}$, but $(\mathbb{R}, +, \times)$ is not ordered by $\sqsubseteq$. If $(Q, \oplus, \otimes)$ is a semiring where $\oplus$ is *selective* (*i.e.* $\forall a, b \in Q, a \oplus b = a \vee a \oplus b = b$), then $(Q, \oplus, \otimes)$ is an ordered semiring, and $\sqsubseteq$ is a total order (*i.e.* $\forall a, b \in Q, a \sqsubseteq b \vee b \sqsubseteq a$). Thus, another example of ordered semirings is $(\mathbb{R} \cup \{+\infty\}, \min, +)$, with $\sqsubseteq$ equal to the $\geqslant$ order over reals. If $(Q, \oplus, \otimes)$ is an ordered semiring, then for every subset $Q' \subseteq Q$ such that $\oplus$ and $\otimes$ are stable over Q' and such that $\mathbf{0}$ and $\mathbf{1}$ are in Q', $(Q', \oplus, \otimes)$ is also an ordered semiring. Therefore, $(\mathbb{N}, +, \times)$ and $(\mathbb{Q} \cup \{+\infty\}, \min, +)$ are ordered semirings.

If $(Q, \oplus, \otimes)$ is an ordered semiring, then $\mathbf{0}$ is the least element of Q w.r.t. $\sqsubseteq$, *i.e.* $\forall a \in Q, \mathbf{0} \sqsubseteq a$. If there exists in Q an absorbing element $\boldsymbol{\infty}$ for $\oplus$ (such that for all $a \in Q$, $\boldsymbol{\infty} \oplus a = \boldsymbol{\infty}$), then $\boldsymbol{\infty}$ is the greatest element of Q w.r.t. $\sqsubseteq$, *i.e.* $\forall a \in Q, a \sqsubseteq \boldsymbol{\infty}$. For example, $(\mathbb{R}_{\geqslant 0} \cup \{+\infty\}, \min, +)$ and $(\{0, 1\}, \vee, \wedge)$ are ordered semirings where $\boldsymbol{\infty}$ equals 0 and 1, respectively. If Q does not contain an absorbing element for $\oplus$, we can consider a new symbol $\boldsymbol{\infty} \notin Q$ and extend $\oplus$ and $\otimes$ with $a \oplus \boldsymbol{\infty} = \boldsymbol{\infty}$ for $a \in Q$,

[1] If $\mathbf{0}$ and $\mathbf{0}'$ are neutral for $\oplus$ then $\mathbf{0} = \mathbf{0} \oplus \mathbf{0}' = \mathbf{0}'$, this also holds on $\mathbf{1}$ with $\otimes$.

$\boldsymbol{\infty} \oplus \boldsymbol{\infty} = \boldsymbol{\infty}$, $a \otimes \boldsymbol{\infty} = \boldsymbol{\infty} \otimes a = \boldsymbol{\infty}$ for $a \in Q \backslash \{\mathbf{0}\}$, $\mathbf{0} \otimes \boldsymbol{\infty} = \boldsymbol{\infty} \otimes \mathbf{0} = \mathbf{0}$ and $\boldsymbol{\infty} \otimes \boldsymbol{\infty} = \boldsymbol{\infty}$, such that $(Q \cup \{\boldsymbol{\infty}\}, \oplus, \otimes, \mathbf{0}, \mathbf{1})$ is an ordered semiring with $\boldsymbol{\infty}$ absorbing for $\oplus$. This allows us to define the ordered semirings $(\mathbb{R}_{\geqslant 0} \cup \{+\infty\}, +, \times)$ with $\boldsymbol{\infty} = +\infty$ and $(\mathbb{R} \cup \{+\infty, -\infty\}, \min, +)$ with $\boldsymbol{\infty} = -\infty$ for example, and in the following we will assume that every ordered semiring contains an absorbing element $\boldsymbol{\infty}$ for $\oplus$.

For every $k \geqslant 0$ and $a \in Q$ in an ordered semiring $(Q, \oplus, \otimes)$, let a^k denote $\mathbf{1}$ if $k = 0$ and $\bigotimes_{0 \leqslant i < k} a$ (*i.e.* $a \otimes a \cdots \otimes a$, k times) if $k > 0$, and let $a^{(k)}$ denote $\bigoplus_{0 \leqslant i \leqslant k} a^i$ (*i.e.* $a^0 \oplus \cdots \oplus a^k$). On ordered semirings, $\oplus$ is monotone (*i.e.* $\forall a, b \in Q, a \sqsubseteq a \oplus b$), and therefore the sequence $(a^{(k)})_{k \in \mathbb{N}}$ is non-decreasing. Let $a^{(*)}$ be called the *closure* of a and denote the supremum of $(a^{(k)})_{k \in \mathbb{N}}$ if it exists. Intuitively, $a^{(*)}$ is the limit of the infinite sum $\mathbf{1} \oplus a \oplus (a \otimes a) \oplus (a \otimes a \otimes a) \oplus \dots$ in Q.

Definition 1.7. An *ordered semiring with closure* is an ordered semiring where the closure $a^{(*)}$ of every $a \in Q$ exists, *i.e.* the set $\mathcal{S}_a = \{b \in Q \mid \forall k \in \mathbb{N}, a^{(k)} \sqsubseteq b\}$ contains an element $a^{(*)}$ such that $a^{(*)} \sqsubseteq b$ for every $b \in \mathcal{S}_a$.

A first class of ordered semirings with closure are those where $\mathbf{1} = \boldsymbol{\infty}$, *i.e.* the neutral element for $\otimes$ is absorbing for $\oplus$. In this case, $a^{(*)} = \mathbf{1}$ for every $a \in Q$. The semirings $(\mathbb{R}_{\geqslant 0} \cup \{+\infty\}, \min, +)$ and $(\{0, 1\}, \vee, \wedge)$ belong to this class. We now introduce another class of ordered semirings with closure, called *complete semirings*.

The relation $\sqsubseteq$ is a complete order over Q if each of its subsets has a supremum, *i.e.* for all subsets S of Q, there exists c in $B = \{b \in Q \mid \forall a \in S, a \sqsubseteq b\}$ such that $c \sqsubseteq b$ for every $b \in B$. In this case, we say that Q equipped with $\sqsubseteq$ is a complete join semilattice, where join is the operator that returns the supremum of a subset of Q. A complete semiring is an ordered semiring $(Q, \oplus, \otimes)$ such that the relation $\sqsubseteq$ is a complete order over Q.

Lemma 1.1. *Complete semirings are ordered semirings with closure.*

Proof. Consider some element $a \in Q$, and let f_a be a unary operator over Q defined by $f_a(q) = (a \otimes q) \oplus \mathbf{1}$. In particular, $f_a(\mathbf{0}) = \mathbf{1} = a^{(0)}$, and for every $k \geqslant 0$, $f_a(a^{(k)}) = a^{(k+1)}$. Let us prove that f_a is non-decreasing, by considering $q_1 \sqsubseteq q_2$ (*i.e.* $\exists c, q_2 = q_1 \oplus c$) and showing $f_a(q_1) \sqsubseteq f_a(q_2)$:

$$f_a(q_2) = f_a(q_1 \oplus c) = (a \otimes (q_1 \oplus c)) \oplus \mathbf{1} = (a \otimes q_1) \oplus (a \otimes c) \oplus \mathbf{1} = f_a(q_1) \oplus (a \otimes c)\,.$$

It follows that on a complete semiring, f_a is Scott-continuous (it preserves the supremum of sets), and by Kleene's fixpoint theorem [SHLG94], f_a has a least fixpoint, which is the supremum of the non-decreasing chain

$$\mathbf{0} \sqsubseteq f_a(\mathbf{0}) \sqsubseteq f_a(f_a(\mathbf{0})) \sqsubseteq f_a(f_a(f_a(\mathbf{0}))) \cdots$$

i.e. the supremum of $(a^{(k)})_{k \in \mathbb{N}}$, and therefore $a^{(*)}$ by definition. □

Example 1.3. Some examples of complete semirings include $(\mathbb{R}_{\geqslant 0} \cup \{+\infty\}, +, \times)$, where $a^{(*)}$ equals $+\infty$ if $a \geqslant 1$ and $1/(1-a)$ if $a < 1$, and $(\mathbb{R} \cup \{-\infty, +\infty\}, \min, +)$ where $a^{(*)}$ equals 0 if $a \geqslant 0$, and $-\infty$ if $a < 0$. Both form complete join semilattices with

$\sqsubseteq$ where join is either the usual sup or inf operator over reals. $(\mathbb{N} \cup \{+\infty\}, +, \times)$ and $(\mathbb{N} \cup \{+\infty\}, \min, +)$ are also complete semirings, but not $(\mathbb{Q} \cup \{-\infty, +\infty\}, \min, +)$ as the ordinary order $\geqslant$ is not complete over rational numbers (the infimum of a subset of $\mathbb{Q}$ may be irrational). However, if we let $\mathbb{Q}_N = \{a/N \mid a \in \mathbb{Z}\}$ be the set of rational numbers of granularity $1/N$ for a fixed $N \in \mathbb{N}_{>0}$, then the semiring $(\mathbb{Q}_N \cup \{-\infty, +\infty\}, \min, +)$ is complete, and the closure operation is inherited from the tropical semiring over $\mathbb{R} \cup \{-\infty, +\infty\}$.

Let us denote $\mathcal{M}_n(Q)$ the set of $n \times n$ matrices over Q. If $(Q, \oplus, \otimes)$ is an ordered semiring, then, for every integer $n > 0$, $(\mathcal{M}_n(Q), \oplus, \otimes, \mathbf{0}, \mathbf{1})$ is an ordered semiring, with $\oplus$ and $\otimes$ the entrywise addition and the standard multplication of matrices (using $\oplus$ and $\otimes$ over Q internally), $\mathbf{0}$ the null matrix (equal to $\mathbf{0}$ everywhere), and $\mathbf{1}$ the identity matrix (equal to $\mathbf{1}$ on the diagonal and to $\mathbf{0}$ everywhere else). The order $\sqsubseteq$ over $\mathcal{M}_n(Q)$ is inherited by applying the order $\sqsubseteq$ over Q entrywise, and for every matrix $A \in \mathcal{M}_n(Q)$, $A^{(*)}$ denotes the closure of A in the matrix semiring (*i.e.* $\mathbf{1} \oplus A \oplus A^2 \oplus A^3 \oplus \dots$) when it exists.

Lemma 1.2 (Generalized Gauss-Jordan, [GM08])**.** *If $(Q, \oplus, \otimes)$ is an ordered semiring with closure, then $A^{(*)}$ exists for every matrix A, so that $(\mathcal{M}_n(Q), \oplus, \otimes)$ is also an ordered semiring with closure. Moreover, $A^{(*)}$ can be computed by performing at most n^3 elementary operations ($\oplus$, $\otimes$, and closure in Q).*

This result is derived from a generalised version of the Floyd-Warshall algorithm defined over semirings, described in Algorithm 1.1. To get intuition on Algorithm 1.1, one can interpret the matrix A as a graph $G = \langle \{1, \dots, n\}, \{i \xrightarrow{M(i,j)} j \mid M(i,j) \neq \mathbf{0}\} \rangle$ labelled over Q, such that the entry (i, j) in $A^{(*)}$ corresponds to applying $\bigoplus$, over all paths ρ from i to j in G, on $\bigotimes_{k \xrightarrow{a} l \in \rho} a$. The closure of diagonal entries corresponds to an acceleration of this computation over cycles, such that only paths without cycles need to be considered. Whenever the variable k is incremented, A_k contains the result of this computation over all paths without cycles that only use $\{1, \dots, k\}$ as intermediate states, and thus $A_n \oplus \mathbf{1} = A^{(*)}$ (that last step only updates diagonal entries by adding $\mathbf{1}$, this corresponds to paths of length zero).

1.2.2. Transition systems labelled over a semiring

The results of this section hold for every semiring, but as we will mostly consider semirings that extend the tropical semiring in further chapters, we will change notations and name the two operators min and + of neutral elements $+\infty$ and 0 instead of $\oplus$ and $\otimes$ of neutral elements $\mathbf{0}$ and $\mathbf{1}$. We will also assume that min has an absorbing element named $-\infty$ instead of $\boldsymbol{\infty}$. In this section, we consider only finite transition systems (finitely many states and edges).

Definition 1.8. A *weighted transition system* is a transition system $\langle S, T \rangle$ labelled over Σ, such that $(\Sigma, \min, +)$ is an ordered semiring with closure.

Algorithm 1.1: Closure computation over the semiring of matrices [GM08]

Input : $A \in \mathcal{M}_n(Q)$
Output : $A^{(*)}$

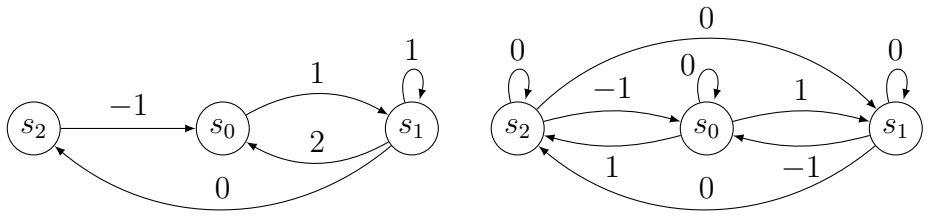
```
/* We construct a sequence of matrices A_0 ... A_n                    */
```
1 $A_0 \leftarrow A$
2 **for** $k \leftarrow 1$ **to** n **do**
3 $\quad A_k(k,k) \leftarrow (A_{k-1}(k,k))^{(*)}$
4 $\quad$ **for** $i \leftarrow 1$ **to** n **do**
5 $\quad\quad$ **for** $j \leftarrow 1$ **to** n **do**
6 $\quad\quad\quad$ **if** $(i,j) \neq (k,k)$ **then**
7 $\quad\quad\quad\quad A_k(i,j) \leftarrow A_{k-1}(i,j) \oplus (A_{k-1}(i,k) \otimes A_k(k,k) \otimes A_{k-1}(k,j))$
8 **return** $A_n \oplus \mathbf{1}$

The weight of a finite path $\rho = s_0 \xrightarrow{a_1} s_1 \dots \xrightarrow{a_k} s_k$ is obtained by summing its edges in order ($+$ may not be commutative) $\mathsf{wt}(\rho) = a_1 + \cdots + a_k$. The weight $\mathsf{wt}^k(s, s')$ for a pair of states (s, s') and $k \in \mathbb{N}$ is defined as the minimal weight of the set of all paths from s to s' of length exactly k in the transition system.[2] We also let $\mathsf{wt}^{\leqslant k}(s, s') = \min_{0 \leqslant i \leqslant k} \mathsf{wt}^i(s, s')$ denote the minimal weight for all paths of length at most $k \in \mathbb{N}$. From the ordering $\sqsubseteq$ of $(\Sigma, \min, +)$ we derive an ordering $\leqslant$, such that $a \leqslant b \Leftrightarrow b \sqsubseteq a \Leftrightarrow \exists c \in \Sigma, a = \min(b, c)$. The sequence $(\mathsf{wt}^{\leqslant k}(s, s'))_{k \in \mathbb{N}}$ is decreasing for $\leqslant$, and we define $\mathsf{wt}(s, s')$ as its infimum w.r.t. $\leqslant$ if it exists. In fact, we will see that it always exists when $(\Sigma, \min, +)$ is a semiring with closure.

Example 1.4. If the semiring is the tropical semiring $(\mathbb{N} \cup \{+\infty\}, \min, +)$, then $\mathsf{wt}(s, s')$ corresponds to the weight of the shortest path from s to s' in a transition system labelled by non-negative weights representing distance, and equals $+\infty$ if no such path exists. If the semiring is $(\mathbb{Z} \cup \{-\infty, +\infty\}, \min, +)$, $\mathsf{wt}(s, s')$ corresponds to the infimum of the weight of the paths from s to s', *i.e.* the weight of the shortest path if it exists, $+\infty$ if s cannot reach s', and $-\infty$ if s can reach a negative cycle that can reach s'. If the semiring is $(\{0, 1\}, \vee, \wedge)$, $\mathsf{wt}(s, s') = 1$ if and only if there is a path from s to s' entirely labelled by 1.

The *adjacency matrix* of a transition system $\langle S, T \rangle$ weighted over Σ is a matrix M in $\mathcal{M}_{|S|}(\Sigma)$, seen as a mapping from $S \times S$ to Σ, such that $M(s, s')$ is equal to $\mathsf{wt}^1(s, s')$. M is an element of the ordered semiring with closure $(\mathcal{M}_{|S|}(\Sigma), \min, +)$, where min is the entrywise application of min and $+$ is the standard product of matrices over $(Q, \min, +)$. Observe that for every pair of states (s, s'), it holds that for every $k \geqslant 0$, $M^{(k)}(s, s') = \mathsf{wt}^{\leqslant k}(s, s')$, and therefore $M^{(*)}(s, s') = \mathsf{wt}(s, s')$. Then, we can compute $\mathsf{wt}(s, s')$ for every pair (s, s') by using Algorithm 1.1 on M. From the adjacency matrix

[2] In particular, $\mathsf{wt}^k(s, s')$ equals $+\infty$ if there are no such path, and $\mathsf{wt}^0(s, s')$ is equal to 0 if $s = s'$ and $+\infty$ otherwise.



$$M = \begin{array}{c} \\ s_0 \\ s_1 \\ s_2 \end{array} \begin{array}{c} \begin{array}{ccc} s_0 & s_1 & s_2 \end{array} \\ \begin{pmatrix} +\infty & 1 & +\infty \\ 2 & 1 & 0 \\ -1 & +\infty & +\infty \end{pmatrix} \end{array} \qquad M^{(*)} = \begin{array}{c} \\ s_0 \\ s_1 \\ s_2 \end{array} \begin{array}{c} \begin{array}{ccc} s_0 & s_1 & s_2 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 \\ -1 & 0 & 0 \\ -1 & 0 & 0 \end{pmatrix} \end{array}$$

Figure 1.2.: A weighted transition system labelled over $(\mathbb{Z} \cup \{+\infty, -\infty\}, \min, +)$ and its adjacency matrix on the left, their closure on the right.

$M^{(*)}$ of a transition system $\langle S, T \rangle$, we can extract a transition system $\langle S, T' \rangle$ called the closure of $\langle S, T \rangle$, with $T' = \{s \xrightarrow{M^{(*)}(s,s')} s' \mid s, s' \in S\}$.

Example 1.5. Figure 1.2 represents a transition system labelled over the tropical semiring $(\mathbb{Z} \cup \{+\infty, -\infty\}, \min, +)$, its adjacency matrix M containing the minimum weight over paths of length 1, the closure $M^{(*)} = \min(\mathbf{1}, M, M^2, M^3)$ ($\mathbf{1}$ is the identity matrix with 0 on the diagonal and $+\infty$ everywhere else), and the transition system associated with $M^{(*)}$.

On $(\mathbb{N} \cup \{+\infty\}, \min, +)$, Algorithm 1.1 is equivalent to running the classical Floyd-Warshall algorithm on a graph with non-negative weights. On the semiring $(\mathbb{Z} \cup \{-\infty, +\infty\}, \min, +)$, Algorithm 1.1 is equivalent to running the Floyd-Warshall algorithm on a graph with arbitrary weights, with an additional check that sets diagonal entries to $-\infty$ as soon as they become negative.

1.3. Turn-based game on a transition system

In order to model situations where some choices are out of our control, we will study a game-theoretical extension of transition systems, where two players play a turn-based game.

Definition 1.9. A *two-player turn-based game* labelled over Σ is a tuple $\mathcal{G} = \langle S_{\mathsf{Ctrl}}, S_{\mathsf{Env}}, T \rangle$ such that $S_{\mathsf{Ctrl}} \cap S_{\mathsf{Env}} = \emptyset$ and $\langle S_{\mathsf{Ctrl}} \cup S_{\mathsf{Env}}, T \rangle$ is a transition system labelled over Σ.

The disjoint union of states is denoted $S = S_{\mathsf{Ctrl}} \uplus S_{\mathsf{Env}}$. We say that S_{Ctrl} contains the states that belong to the player Ctrl, and S_{Env} the states that belong to the player Env. A maximal play (resp. a non-maximal play) ρ in $\mathcal{G}$ is a maximal path (resp. a non-maximal path) in $\langle S, T \rangle$. Let $S_\perp \subseteq S$ denote the deadlock states. Recall that states can be seen as paths of length 0, and therefore states in $S \backslash S_\perp$ can be seen as non-maximal plays. For $\mathsf{P} \in \{\mathsf{Ctrl}, \mathsf{Env}\}$, the set of non-maximal plays ρ such that $\mathsf{last}(\rho) \in S_{\mathsf{P}}$ is denoted

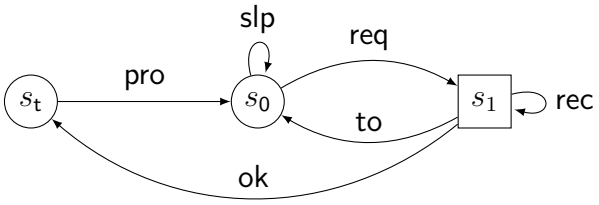


Figure 1.3.: A two-player turn-based game, where controller owns the circle states s_0 and s_t, and the environment owns the rectangle state s_1.

$\mathsf{FPlays}^{\mathsf{P}}$. A strategy σ_{P} for player P is a mapping from $\mathsf{FPlays}^{\mathsf{P}}$ to T, such that for all $\rho \in \mathsf{FPlays}^{\mathsf{P}}$, $\mathsf{last}(\rho) = \mathsf{first}(\sigma_{\mathsf{P}}(\rho))$. A strategy is said positional if for all $\rho \in \mathsf{FPlays}^{\mathsf{P}}$, $\sigma_{\mathsf{P}}(\rho) = \sigma_{\mathsf{P}}(\mathsf{last}(\rho))$.[3] Let $\mathsf{play}(s_0, \sigma_{\mathsf{Ctrl}}, \sigma_{\mathsf{Env}})$ denote the unique maximal play obtained from an initial state and a pair of strategies, such that $\mathsf{first}(\mathsf{play}(s_0, \sigma_{\mathsf{Ctrl}}, \sigma_{\mathsf{Env}})) = s_0$, and for every prefix ρ of $\mathsf{play}(s_0, \sigma_{\mathsf{Ctrl}}, \sigma_{\mathsf{Env}})$ in $\mathsf{FPlays}^{\mathsf{P}}$, the next transition in $\mathsf{play}(s_0, \sigma_{\mathsf{Ctrl}}, \sigma_{\mathsf{Env}})$ is $\sigma_{\mathsf{P}}(\rho)$.

Definition 1.10. Given a game $\mathcal{G}$, an objective $\mathcal{L}_t$ (as an ω-language over labels) and an initial state s_0, the *controller synthesis problem* asks if there exists a strategy σ_{Ctrl} for Ctrl such that for all strategies σ_{Env} for Env, $\mathsf{play}(s_0, \sigma_{\mathsf{Ctrl}}, \sigma_{\mathsf{Env}})$ is an infinite play that induces a word in $\mathcal{L}_t$.

Example 1.6. Figure 1.3 represents a two-player turn-based game associated with the transition system of Figure 1.1, such that $S_{\mathsf{Ctrl}} = \{s_0, s_t\}$ and $S_{\mathsf{Env}} = \{s_1\}$. This models the fact that, as a client, we do not control the server's answer. On this example, consider the reachability objective associated with $S_t = \{s_t\}$, that models the specification "at least one communication goes well". The synthesis problem is not satisfied from s_0 with this objective, since for every strategy of player Ctrl, player Env can choose to always time-out, or loop in s_1 forever. If we consider the specification "controller sends at least one request", defined by the reachability objective associated with $S_t = \{s_1\}$, the synthesis problem is satisfied from s_0, for example with the positional strategy for controller that chooses req in s_0 and pro in s_t.

If S_{Env} is empty, the controller synthesis problem on $\mathcal{G}$ is equivalent to the emptiness problem on the transition system $\langle S_{\mathsf{Ctrl}}, T\rangle$. If S_{Ctrl} is empty, the controller synthesis problem on $\mathcal{G}$ is equivalent to the model-checking problem on the transition system $\langle S_{\mathsf{Env}}, T\rangle$.

For a finite system equipped with a reachability or Büchi objective, the controller synthesis problem is polynomial, and can be solved with fixpoint computations. In a more general setting where the objective is given as an LTL formula, the controller synthesis problem is 2-EXPTIME complete [PR89].

[3] Positional strategies are often called *memoryless* in the literature, as they always make the same decision in a given state, thus one does not need to remember the history ρ to follow them.

1.3.1. Attractors

We now recall the classical notion of attractor of a player towards a set of states, that is used to solve reachability games. Let $S_{\mathsf{t}} \subseteq S$ be a set of states. The attractor of Ctrl towards S_{t} is the set of states such that player Ctrl can guarantee reaching S_{t} eventually. Formally, it is the greatest set $S' \subseteq S$ such that for all $s \in S'$, either:

1. $s \in S_{\mathsf{t}}$; or
2. $s \in S_{\mathsf{Ctrl}}$, and there exists a transition $s \xrightarrow{a} s'$ with $s' \in S'$; or
3. $s \in S_{\mathsf{Env}}$, and for all transitions $s \xrightarrow{a} s'$, it holds that $s' \in S'$.

It is well-known that S' can be computed with a fixpoint computation that starts with $S' = S_{\mathsf{t}}$, and adds progressively the states that satisfy conditions 2 or 3, until no such state is left. The complexity of this computation is linear in the size $|S| + |T|$ of the graph.[4]

Then, the controller synthesis problem with reachability objective S_{t} and initial state s_0 is satisfied if and only if s_0 belongs to the attractor of Ctrl towards S_{t}, and in this case one can extract from the attractor computation a (positional) strategy σ_{Ctrl} that guarantees reachability of S_{t} from s_0.

A symmetrical notion of attractor of Env towards a set S_{t} can be defined and computed similarly.

[4] A single backwards breadth-first search from S_{t} is enough if one keeps track, for each state of Max, of the number of successors that have not yet been added to S'. When this counter reaches 0 the state is added.

2. Timed systems

In this chapter, we introduce notions that let us express timing constraints, define timed automata as finite transition systems enriched by those notions, and introduce classical tools for their study.

2.1. Modelling real-time constraints

Let $\mathcal{X} = \{\mathsf{x}_1, \ldots, \mathsf{x}_n\}$ be a finite, non-empty set of variables called clocks. A *valuation* $\nu \colon \mathcal{X} \to \mathbb{R}_{\geqslant 0}$ is a mapping from clocks to non-negative real numbers, such that $\nu(\mathsf{x}_1), \ldots, \nu(\mathsf{x}_n)$ are called the coordinates of ν. Equivalently, ν can be seen as a point in space $\mathbb{R}^{\mathcal{X}}_{\geqslant 0}$. We denote $\mathbf{0}$ the valuation such that for all $\mathsf{x} \in \mathcal{X}$, $\nu(\mathsf{x}) = 0$. Given a real number $d \in \mathbb{R}$, we define $\nu + d$ as the valuation such that $\forall \mathsf{x} \in \mathcal{X}, (\nu + d)(\mathsf{x}) = \nu(\mathsf{x}) + d$ if it exists.[1] If d is non-negative, we say that we performed a time elapse of delay d. The time-successors of ν are the valuations $\nu + d$ with $d \geqslant 0$. Similarly, we refer to all $\nu + d$ in $\mathbb{R}^{\mathcal{X}}_{\geqslant 0}$ with $d \leqslant 0$ as time-predecessors of ν. The set of points that are either time-predecessors or time-successors of a valuation ν form the unique diagonal line in $\mathbb{R}^{\mathcal{X}}_{\geqslant 0}$ that contains ν. If $\mathcal{Y}$ is a subset of $\mathcal{X}$, we define $\nu[\mathcal{Y} := 0]$ as the valuation such that $\forall \mathsf{x} \in \mathcal{Y}, (\nu[\mathcal{Y} := 0])(\mathsf{x}) = 0$ and $\forall \mathsf{x} \in \mathcal{X} \backslash \mathcal{Y}, (\nu[\mathcal{Y} := 0])(\mathsf{x}) = \nu(\mathsf{x})$. This operation is called a reset of clocks $\mathcal{Y}$.

We extend those notions to sets of valuations in a natural way. The set of time-successors of $Z \subseteq \mathbb{R}^{\mathcal{X}}_{\geqslant 0}$, denoted $\mathsf{PostTime}(Z)$, contains the valuations that are time-successors of valuations in Z. The reset of $Z \subseteq \mathbb{R}^{\mathcal{X}}_{\geqslant 0}$ by $\mathcal{Y}$, denoted $Z[\mathcal{Y} := 0]$, contains the valuations $\nu[\mathcal{Y} := 0]$ such that $\nu \in Z$.

The term *atomic constraint* will refer to a linear inequality in one of the following forms:

- A strict (resp. non-strict) *non-diagonal* atomic constraint over clock $\mathsf{x} \in \mathcal{X}$ and constant $c \in \mathbb{Q}$ is an inequality of the form $\mathsf{x} \bowtie c$ with $\bowtie \in \{>, <\}$ (resp. $\bowtie \in \{\geqslant, \leqslant\}$).
- A strict (resp. non-strict) *diagonal* atomic constraint over clocks x and $\mathsf{y} \in \mathcal{X}$ and constant $c \in \mathbb{Q}$ is an inequality of the form $\mathsf{x} - \mathsf{y} \bowtie c$ with $\bowtie \in \{>, <\}$ (resp. $\bowtie \in \{\geqslant, \leqslant\}$).

Let $\top$ and $\bot$ denote two special atomic constraints, defined as $\mathsf{x} \geqslant 0$ and $\mathsf{x} < 0$ for an arbitrary $\mathsf{x} \in \mathcal{X}$. A *guard* g over $\mathcal{X}$ is a finite conjunction of atomic constraints over clocks in $\mathcal{X}$. In particular, guards let us define $\mathsf{x} = c$ as shorthand for $\mathsf{x} \leqslant c \wedge \mathsf{x} \geqslant c$, and

[1] if d is negative, $\nu + d$ may not belong to $\mathbb{R}^{\mathcal{X}}_{\geqslant 0}$

$c_1 < \mathsf{x} < c_2$ as shorthand for $\mathsf{x} > c_1 \wedge \mathsf{x} < c_2$. A guard is said strict (resp. non-strict, diagonal, non-diagonal) if all of its atomic constraints are strict (resp. non-strict, diagonal, non-diagonal). $\mathsf{Guards}(\mathcal{X})$ denotes the set of all guards over $\mathcal{X}$, and $\mathsf{Guards}^{\mathrm{nd}}(\mathcal{X})$ the subset of non-diagonal guards. For all constants $c \in \mathbb{Q}$ and $\bowtie \in \{\geqslant, \leqslant, >, <\}$, we say that valuation $\nu \in \mathbb{R}^{\mathcal{X}}_{\geqslant 0}$ satisfies the atomic constraint $\mathsf{x} \bowtie c$ (resp. $\mathsf{x} - \mathsf{y} \bowtie c$), and write $\nu \models \mathsf{x} \bowtie c$ (resp. $\nu \models \mathsf{x} - \mathsf{y} \bowtie c$), if $\nu(\mathsf{x}) \bowtie c$ (resp. $\nu(\mathsf{x}) - \nu(\mathsf{y}) \bowtie c$). We say that valuation $\nu \in \mathbb{R}^{\mathcal{X}}_{\geqslant 0}$ satisfies guard g, and write $\nu \models g$, if ν satisfies all atomic constraints in g. For $g \in \mathsf{Guards}(\mathcal{X})$, let $[\![g]\!]$ denote the set of all $\nu \in \mathbb{R}^{\mathcal{X}}_{\geqslant 0}$ such that $\nu \models g$. Such sets are called *zones* and form convex polyhedra of $\mathbb{R}^{\mathcal{X}}_{\geqslant 0}$. A guard g is said satisfiable when the zone $[\![g]\!]$ is non-empty, and a zone is called rectangular when the associated guard is non-diagonal. The universal zone refers to $[\![\top]\!] = \mathbb{R}^{\mathcal{X}}_{\geqslant 0}$ and the empty zone refers to $[\![\bot]\!] = \emptyset$. Guard $\overline{g}$ is the closed version of a satisfiable guard g where every strict constraint of comparison operator $<$ or $>$ is replaced by its non-strict version $\leqslant$ or $\geqslant$. The zone $[\![\overline{g}]\!]$ is the topological closure of $Z = [\![g]\!]$, and is also denoted $\overline{Z}$.

Zones are closed by intersection as $[\![g]\!] \cap [\![g']\!] = [\![g \wedge g']\!]$, but not by union.[2] Zones are closed by time-successors as $\mathsf{PostTime}([\![g]\!])$ is equal to $[\![g']\!]$, where g' is obtained from g by removing every non-diagonal atomic constraint of the form $\mathsf{x} < c$ or $\mathsf{x} \leqslant c$. Zones are also closed by reset of clocks $\mathcal{Y} \subseteq \mathcal{X}$, as $[\![g]\!]\,[\mathcal{Y} := 0] = [\![g']\!]$, where $g' = \bot$ if $[\![g]\!] = \emptyset$, and otherwise g' is obtained from g by removing every non-diagonal atomic constraint of the form $\mathsf{x} > c$ or $\mathsf{x} \geqslant c$ with $\mathsf{x} \in \mathcal{Y}$, replacing every diagonal atomic constraint of the form $\mathsf{x} - \mathsf{y} \bowtie c$ with $\mathsf{y} \in \mathcal{Y}$ (resp. $\mathsf{x} \in \mathcal{Y}$) by $\mathsf{x} \bowtie c$ (resp. $\mathsf{y} \not\bowtie -c$), and adding the constraint $\mathsf{x} \leqslant 0$ for every $\mathsf{x} \in \mathcal{Y}$. Note that encoding zones by storing their associated guard syntactically as a formula is not efficient, as guards can contain useless constraints. Moreover, it is possible to have different guards associated to the same zone, and for example testing if a zone is equal to another zone is non-trivial in this form.

2.2. Encoding constraints as DBMs

A bound (over $\mathbb{R}$) is a pair $(\prec, c)$ with $\prec \in \{<, \leqslant\}$ and $c \in \mathbb{R}$. It represents the (open or large) upper bound $\prec c$ in a linear inequality. We introduce additional bounds $(<, +\infty)$ and $(<, -\infty)$ that will be used for trivial inequalities, and let $\mathsf{Bounds}(\mathbb{R})$ denote $(\{<, \leqslant\} \times \mathbb{R}) \cup \{(<, +\infty), (<, -\infty)\}$.

We say that a real number $a \in \mathbb{R}$ satisfies $(\prec, c) \in \{<, \leqslant\} \times \mathbb{R}$ if the inequality $a \prec c$ holds. The bound $(<, +\infty)$ is satisfied by every real and $(<, -\infty)$ is never satisfied. We say that bound $(\prec, c)$ is at least as strong as $(\prec', c')$, denoted $(\prec, c) \preccurlyeq (\prec', c')$, if all reals satisfying $(\prec, c)$ satisfy $(\prec', c')$, equivalently if

$$c = -\infty \vee c' = +\infty \vee c < c' \vee (c = c' \wedge (\prec = \prec' \vee \prec = <)) .$$

This forms a total order over bounds, where the strongest bound is $(<, -\infty)$ and the weakest one is $(<, +\infty)$. We define a binary operator min over bounds that returns the strongest bound out of its arguments, and an infimum inf over sets of bounds that

[2] the union of two zones may not be a convex set

returns the weakest bound that is at least as strong as every bound in the set.

We also define an addition $+$ such that

$$(\prec, c) + (\prec', c') = (\prec'', c + c')\,,$$

with $\prec''$ set to $\leqslant$ if $\prec = \prec' = \leqslant$ and to $<$ otherwise, and where $c + c'$ uses the $+$ operator of the tropical semiring $(\mathbb{R} \cup \{-\infty, +\infty\}, \min, +)$.[3]

The min operation admits $(<, +\infty)$ as neutral element and $(<, -\infty)$ as absorbing element, and $+$ admits $(\leqslant, 0)$ as neutral element and $(<, +\infty)$ as absorbing element. Then, $(\mathsf{Bounds}(\mathbb{R}), \min, +)$ is a semiring that we call the *tropical semiring of bounds* over $\mathbb{R}$. It is an ordered semiring because min is selective. The order $\sqsubseteq$ induced by min in $(\mathsf{Bounds}(\mathbb{R}), \min, +)$ is $\{(a, b) \mid b \preccurlyeq a\}$. Then, $(\mathsf{Bounds}(\mathbb{R}), \min, +)$ forms a complete join semilattice with $\sqsubseteq$ where join is the inf operator. Therefore, $(\mathsf{Bounds}(\mathbb{R}), \min, +)$ is an ordered semiring with closure. The closure of a bound $(\prec, c)$ is equal to: $(<, 0)$ if $(\prec, c) = (<, 0)$; $(<, -\infty)$ if $c < 0$; and $(\leqslant, 0)$ otherwise (*i.e.* if $(\leqslant, 0) \preccurlyeq (\prec, c)$).

We define bounds over a subset Q of $\mathbb{R}$ with $\mathsf{Bounds}(Q) = (\{<, \leqslant\} \times Q) \cup \{(<, +\infty), (<, -\infty)\}$ in a similar fashion. Recall that $\mathbb{Q}_N$ denotes the rational numbers of granularity $1/N$ with $N \in \mathbb{N}_{>0}$, and consider the tropical semiring of bounds over $\mathbb{Q}_N$, defined as $(\mathsf{Bounds}(\mathbb{Q}_N), \min, +)$. It inherits the properties of $(\mathsf{Bounds}(\mathbb{R}), \min, +)$ and is an ordered semiring with closure.

For notational convenience, we add a variable x_0 that is always considered equal to 0, and denote $\mathcal{X}_0$ the set $\mathcal{X} \cup \{\mathsf{x}_0\}$. A difference bound constraint is a linear inequality $\mathsf{x} - \mathsf{y} \prec c$ over clocks $\mathsf{x}, \mathsf{y} \in \mathcal{X}_0$ and bound $(\prec, c)$. As for atomic constraints, we say that a valuation $\nu \in \mathbb{R}_{\geqslant 0}^{\mathcal{X}}$ satisfies $\mathsf{x} - \mathsf{y} \prec c$ and write $\nu \models \mathsf{x} - \mathsf{y} \prec c$ if the real number $\nu(\mathsf{x}) - \nu(\mathsf{y})$ satisfies the bound $(\prec, c)$ (with $\nu(\mathsf{x}_0)$ defined as equal to 0), and $[\![\mathsf{x} - \mathsf{y} \prec c]\!]$ denotes the set of valuations that satisfy $\mathsf{x} - \mathsf{y} \prec c$. The inclusion relation of sets of valuations in $\mathbb{R}_{\geqslant 0}^{\mathcal{X}}$ gives a natural order over atomic constraints, difference bound constraints and guards. Thus, given atomic constraints, difference bound constraints or guards ϕ and ϕ', we will say that ϕ is at least as strong as ϕ' if $[\![\phi]\!] \subseteq [\![\phi']\!]$, and that ϕ is equivalent to ϕ' if $[\![\phi]\!] = [\![\phi']\!]$. A difference bound constraint $\mathsf{x} - \mathsf{y} \prec c$ is at least as strong as another difference bound constraint $\mathsf{x}' - \mathsf{y}' \prec' c'$ if and only if either $(\prec, c) = (<, -\infty)$, $(\prec', c') = (<, +\infty)$, or $\mathsf{x} = \mathsf{x}'$, $\mathsf{y} = \mathsf{y}'$ and $(\prec, c) \preccurlyeq (\prec', c')$.

Lemma 2.1. *Every atomic constraint can be associated with an equivalent difference bound constraint. Similarly, every difference bound constraint can be associated with an equivalent atomic constraint.*

Proof. From atomic constraints to difference bound constraints we refer to Table 2.1. From difference bound constraints to atomic constraints, note that if $c \in \mathbb{Q}$ and at most one of x, y is equal to x_0 then we can use Table 2.1 to find an atomic constraint equivalent to $\mathsf{x} - \mathsf{y} \prec c$. If $c \in \{+\infty, -\infty\}$ we can use the atomic constraint $\top$ or $\bot$. If $\mathsf{x} = \mathsf{y} = \mathsf{x}_0$ then $\mathsf{x}_0 - \mathsf{x}_0 \prec c$ is equivalent to $\top$ if $(\leqslant, 0) \preccurlyeq (\prec, c)$, and to $\bot$ otherwise. □

[3] It is the standard addition over $\mathbb{R}$, extended with $c + (+\infty) = +\infty$ and $c + (-\infty) = -\infty$ for all $c \in \mathbb{R}$, and $(+\infty) + (-\infty) = +\infty$.

$\mathsf{x} \prec c$	$\mathsf{x} \succ c$	$\mathsf{x} - \mathsf{y} \prec c$	$\mathsf{x} - \mathsf{y} \succ c$
$\mathsf{x} - \mathsf{x}_0 \prec c$	$\mathsf{x}_0 - \mathsf{x} \prec -c$	$\mathsf{x} - \mathsf{y} \prec c$	$\mathsf{y} - \mathsf{x} \prec -c$

Table 2.1.: The first line represents atomic constraints, and the second line represents their equivalent difference bound constraints. The symbol $\prec$ (resp. $\succ$) can be interpreted as either $<$ or $\leqslant$ (resp. either $>$ or $\geqslant$).

One can then see a guard g as a finite conjunction of difference bound constraints, and encode it as a finite weighted transition system $\langle \mathcal{X}_0, T \rangle$ labelled over $\mathsf{Bounds}(\mathbb{Q})$, whose vertices are clocks in $\mathcal{X}_0$ and such that every transition $\mathsf{x} \xrightarrow{(\prec,c)} \mathsf{y}$ corresponds to a difference bound constraint $\mathsf{x} - \mathsf{y} \prec c$ in g. We also add transitions $\mathsf{x}_0 \xrightarrow{(\leqslant,0)} \mathsf{x}$ for every $\mathsf{x} \in \mathcal{X}_0$ to enforce the non-negativity of clocks. As there are finitely many constraints in g, there exists a granularity $1/N$ such that $(\mathcal{X}_0, T)$ is in fact labelled over $\mathsf{Bounds}(\mathbb{Q}_N)$. The adjacency matrix of such weighted transition system, as defined in Section 1.2.2, is a mapping M from $\mathcal{X}_0 \times \mathcal{X}_0$ to $\mathsf{Bounds}(\mathbb{Q}_N)$ called a *Difference Bound Matrix* (DBM) where every entry $\mathsf{M}(\mathsf{x}, \mathsf{y}) = (\prec, c)$ represents the difference bound constraint $\mathsf{x} - \mathsf{y} \prec c$, and $[\![\mathsf{M}]\!]$ is the zone associated to the conjunction of those constraints.

DBMs were introduced in [BM83, Dil90] for analyzing timed automata, they offer a practical way to represent zones, and one can perform operations like intersections, resets or time-successor computations on DBMs in time quadratic in $|\mathcal{X}|$. We refer to [BY04] for details.[4] A DBM is said in *normal form* if $\mathsf{M} = \mathsf{M}^{(*)}$ in the semiring of matrices over the tropical semiring of bounds. Thus, for every DBM M there exists a unique DBM in normal form $\mathsf{Norm}(\mathsf{M})$ such that $[\![\mathsf{M}]\!] = [\![\mathsf{Norm}(\mathsf{M})]\!]$. One can use Algorithm 1.1 to compute $\mathsf{Norm}(\mathsf{M}) = \mathsf{M}^{(*)}$ in time cubic in $|\mathcal{X}|$. Intuitively, a DBM in normal form contains all of the strongest atomic constraints that are satisfied by its associated zone.

A DBM in normal form M is non-empty (*i.e.* $[\![\mathsf{M}]\!] \neq \emptyset$) if and only if none of its diagonal entries are negative, *i.e.* $\forall \mathsf{x} \in \mathcal{X}_0, (\leqslant, 0) \preccurlyeq \mathsf{M}(\mathsf{x}, \mathsf{x})$. Given two non-empty DBMs in normal form M and M', we have $[\![\mathsf{M}]\!] = [\![\mathsf{M}']\!]$ if and only if $\mathsf{M} = \mathsf{M}'$, and $[\![\mathsf{M}]\!] \subseteq [\![\mathsf{M}']\!]$ if and only if $\mathsf{M}(\mathsf{x}, \mathsf{y}) \preccurlyeq \mathsf{M}'(\mathsf{x}, \mathsf{y})$ for all clocks $\mathsf{x}, \mathsf{y} \in \mathcal{X}_0$. Therefore, testing DBMs in normal form for equality or inclusion of their associated zones requires a time complexity at most quadratic in $|\mathcal{X}|$.

Example 2.1. An example of satisfiable guard, zone and DBM in normal form is displayed in Figure 2.1. Consider the guard over $\mathcal{X} = \{\mathsf{x}_1, \mathsf{x}_2\}$ defined by $g = \mathsf{x}_1 - \mathsf{x}_2 < 1 \wedge \mathsf{x}_1 - \mathsf{x}_2 \geqslant 1$. It is not satisfiable, and $[\![g]\!] = \emptyset$. From g we obtain the DBM $\mathsf{M} = \begin{pmatrix} \leqslant 0 & < +\infty & < +\infty \\ \leqslant 0 & < +\infty & < 1 \\ \leqslant 0 & \leqslant -1 & < +\infty \end{pmatrix}$, with $\mathsf{Norm}(\mathsf{M}) = \begin{pmatrix} \leqslant 0 & < +\infty & < +\infty \\ \leqslant 0 & < 0 & < 1 \\ \leqslant 0 & < -1 & < 0 \end{pmatrix}$.

[4] DBMs are usually defined without the bound $(<, -\infty)$: they only appear when the associated zone is empty, in which case computations are stopped as soon as possible. The two definitions are equivalent for non-empty zones.

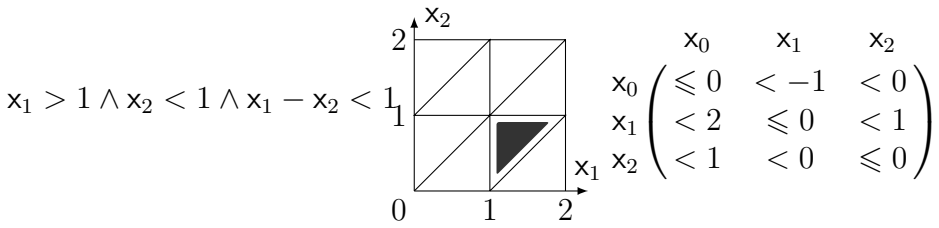


Figure 2.1.: A guard g over clocks $\mathcal{X} = \{\mathsf{x}_1, \mathsf{x}_2\}$ with constants in $\mathbb{Z} = \mathbb{Q}_1$, a representation of the zone $[\![g]\!]$, and the DBM in normal form M encoding $[\![g]\!]$.

2.3. Timed automata

Definition 2.1. A *timed automaton* $\mathcal{A}$ is a tuple $\langle L, \mathcal{X}, \Sigma, E\rangle$, with L a finite set of locations, $\mathcal{X}$ a finite set of clocks, Σ a finite set of actions and E a finite set of edges labelled by an action, a non-diagonal guard and a set of clocks to reset, such that $\langle L, E\rangle$ is a finite transition system labelled over $\Sigma \times \mathsf{Guards}^{\mathrm{nd}}(\mathcal{X}) \times 2^{\mathcal{X}}$.

A path π in $\mathcal{A}$ refers to a path in the transition system $\langle L, E\rangle$, *i.e.* a sequence of the form $\ell_0 \xrightarrow{a_1,g_1,\mathcal{Y}_1} \ell_1 \xrightarrow{a_2,g_2,\mathcal{Y}_2} \dots$ in (L, E). We call a pair $(\ell, \nu) \in L \times \mathbb{R}_{\geqslant 0}^{\mathcal{X}}$ a configuration. The semantics of $\mathcal{A}$ is defined as the transition system $[\![\mathcal{A}]\!] = \langle L \times \mathbb{R}_{\geqslant 0}^{\mathcal{X}}, T\rangle$ labelled over $\mathbb{R}_{\geqslant 0} \times E$, whose states are configurations and where T is the set of transitions $(\ell, \nu) \xrightarrow{d,e} (\ell', \nu')$ obtained from every delay $d \in \mathbb{R}_{\geqslant 0}$ and edge $e = \ell \xrightarrow{a,g,\mathcal{Y}} \ell'$ in $\mathcal{A}$ such that $\nu + d \models g$ and $\nu' = (\nu + d)[\mathcal{Y} := 0]$. An execution ρ in $\mathcal{A}$ refers to a path in $[\![\mathcal{A}]\!]$, it describes a sequence of delays and edges to take, starting from configuration $\mathsf{first}(\rho)$. Given an execution ρ in $\mathcal{A}$, one can abstract delays and valuations to obtain a path π in $\mathcal{A}$. In this case, we say that ρ is following π, or that π contains the execution ρ.

A finite or infinite word $a_1 a_2 \dots$ over Σ is called an *untimed word*. We define *timed words* as finite or infinite words over $\mathbb{R}_{\geqslant 0} \times \Sigma$, such that $(d_1, a_1)(d_2, a_2) \dots$ represents the sequence where action a_1 happens after a delay of d_1 units of time, action a_2 happens d_2 units of time after a_1, *etc.*.[5] A path π in $\mathcal{A}$ naturally induces an untimed word, and an execution ρ in $\mathcal{A}$ naturally induces a timed word. Sets of infinite timed (resp. untimed) words are called timed (resp. untimed) ω-languages. Given a timed ω-language $\mathcal{L}$, we define the corresponding untimed ω-language $\mathsf{Untimed}(\mathcal{L})$ as $\{(a_i)_{i\in\mathbb{N}} \mid \exists (d_i)_{i\in\mathbb{N}} \in \mathbb{R}_{\geqslant 0}^{\mathbb{N}}, (d_i, a_i)_{i\in\mathbb{N}} \in \mathcal{L}\}$, and given an untimed ω-language $\mathcal{L}^u$, we define the corresponding timed ω-language $\mathsf{Timed}(\mathcal{L}^u)$ as $\{(d_i, a_i)_{i\in\mathbb{N}} \mid (d_i)_{i\in\mathbb{N}} \in \mathbb{R}_{\geqslant 0}^{\mathbb{N}}, (a_i)_{i\in\mathbb{N}} \in \mathcal{L}^u\}$.

Example 2.2. An example of timed automaton $\mathcal{A}$ is depicted in Figure 2.2. We denote valuation $\nu : \mathsf{x}_1 \to \nu(\mathsf{x}_1), \mathsf{x}_2 \to \nu(\mathsf{x}_2)$ as $\nu = (\nu(\mathsf{x}_1), \nu(\mathsf{x}_2))$. From $(\ell_0, (0, 0))$, $\mathcal{A}$ allows the finite timed word $(1.5, a)(0.25, b)(1.75, a)$, induced by an execution following $\ell_0 \xrightarrow{a,1<\mathsf{x}_1<2,\{\mathsf{x}_2\}} \ell_1 \xrightarrow{b,\mathsf{x}_1\leqslant 2,\{\mathsf{x}_1\}} \ell_2 \xrightarrow{a,\mathsf{x}_2\geqslant 2,\{\mathsf{x}_2\}} \ell_1$ with successive configurations $(\ell_0, (0, 0))$, $(\ell_1, (1.5, 0))$, $(\ell_2, (0, 0.25))$ and $(\ell_1, (1.75, 0))$. The untimed word ba cannot be induced by an execution starting from $(\ell_0, (0, 0))$.

[5] Timed words are usually defined with global time stamps instead of delays, but one can convert from one to the other as needed.

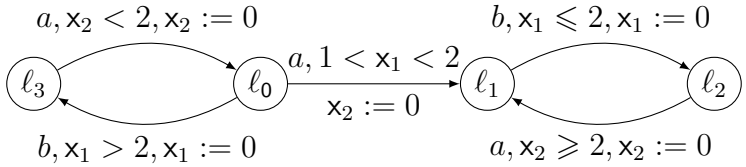


Figure 2.2.: A timed automaton over clocks $\mathcal{X} = \{\mathsf{x}_1, \mathsf{x}_2\}$ and actions $\Sigma = \{a, b\}$, where the reset of a clock x is denoted by $\mathsf{x} := 0$, such that the edge from ℓ_3 to ℓ_0 refers to $(\ell_3, a, \mathsf{x}_2 < 2, \{\mathsf{x}_2\}, \ell_0) \in E$, alternatively denoted $\ell_3 \xrightarrow{a,\mathsf{x}_2<2,\{\mathsf{x}_2\}} \ell_0$.

We define the emptiness and model checking problems on timed automata using the transition system of their semantics. Given a timed automaton $\mathcal{A}$, an objective $\mathcal{L}^u_{\mathsf{t}}$ (as an untimed ω-language), an initial location ℓ_0 and an initial valuation ν_0, the *emptiness problem* (resp. *model-checking problem*) refers to the emptiness (resp. model-checking) problem on $[\![\mathcal{A}]\!]$ with objective $\mathsf{Timed}(\mathcal{L}^u_{\mathsf{t}})$ and initial configuration (ℓ_0, ν_0).

Remark. A standard definition of those problems would use a timed language as the objective, but we will focus on cases where the objective is untimed. It is usually assumed that all clocks are null initially (*i.e.* $\nu_0 = \mathbf{0}$), and in this case only the initial location ℓ_0 is specified.

Given a timed automaton $\mathcal{A}$ and an initial configuration (ℓ_0, ν_0), the timed language $\mathcal{L}_{(\ell_0,\nu_0)}(\mathcal{A})$ is defined as the language of $[\![\mathcal{A}]\!]$ at (ℓ_0, ν_0), *i.e.* the set of timed words induced by the executions starting from (ℓ_0, ν_0). The untimed language $\mathcal{L}^u_{(\ell_0,\nu_0)}(\mathcal{A})$ is defined as $\mathsf{Untimed}(\mathcal{L}_{(\ell_0,\nu_0)}(\mathcal{A}))$, *i.e.* the set of untimed words induced by paths that contain executions starting from (ℓ_0, ν_0).

The emptiness (resp. model-checking) problem can be reformulated as asking if the untimed language $\mathcal{L}^u_{(\ell_0,\nu_0)}(\mathcal{A})$ has non-empty intersection with $\mathcal{L}^u_{\mathsf{t}}$ (resp. is included in $\mathcal{L}^u_{\mathsf{t}}$). Thus, if two timed automata have the same untimed language, they are equivalent for the emptiness and model-checking problems. We also define the emptiness problem with a reachability condition (resp. with a Büchi condition) on timed automata, and they can be seen as special cases of the emptiness problem associated to reachability (resp. Büchi) objectives. Given a timed automaton, an initial configuration (ℓ_0, ν_0), and a set of target locations $L_{\mathsf{t}} \subseteq L$, the *emptiness problem with a reachability condition* asks if there exists a finite execution ρ such that $\mathsf{first}(\rho) = (\ell_0, \nu_0)$ and $\mathsf{last}(\rho) \in L_{\mathsf{t}} \times \mathbb{R}^{\mathcal{X}}_{\geqslant 0}$. Given a timed automaton, an initial configuration (ℓ_0, ν_0), and a set of target locations $L_{\mathsf{t}} \subseteq L$, the *emptiness problem with a Büchi condition* asks if there exists an infinite execution $(\ell_0, \nu_0) \xrightarrow{d_1,e_1} (\ell_1, \nu_1) \xrightarrow{d_2,e_2} \dots$ starting from (ℓ_0, ν_0) that reaches L_{t} infinitely often, *i.e.* such that there exist infinitely many $i \in \mathbb{N}$ with $\ell_i \in L_{\mathsf{t}}$.

2.3.1. Bounded clocks

Let $\mathbb{R}^{\mathcal{X}}_{\geqslant 0,<M}$ denote the set of valuations in $\mathbb{R}^{\mathcal{X}}_{\geqslant 0}$ bounded by $M \in \mathbb{Q}_{\geqslant 0}$, *i.e.* such that for every clock $\mathsf{x} \in \mathcal{X}$, $\nu(\mathsf{x}) \in [0, M)$. A guard g is said to be bounded by $M \in \mathbb{Q}_{\geqslant 0}$ if $[\![g]\!] \subseteq \mathbb{R}^{\mathcal{X}}_{\geqslant 0,<M}$, *i.e.* every valuation that satisfies g belongs to $\mathbb{R}^{\mathcal{X}}_{\geqslant 0,<M}$. Let $\mathsf{Guards}(\mathcal{X}, M)$

(resp. $\mathsf{Guards}^{\mathrm{nd}}(\mathcal{X}, M)$) denote the set of guards (resp. non-diagonal guards) over $\mathcal{X}$ bounded by M. Given a timed automaton $\mathcal{A}$, we say that all clocks are bounded by M in $\mathcal{A}$ if every guard in $\mathcal{A}$ belongs to $\mathsf{Guards}^{\mathrm{nd}}(\mathcal{X}, M)$. Let $\mathcal{A}$ be a timed automaton, and let M be the greatest constant (in absolute value) to appear in the constraints of $\mathcal{A}$. There exists a timed automaton $\mathcal{A}'$ where clocks are bounded by $M+1$, and an initial location ℓ_0' in $\mathcal{A}'$ such that $\mathcal{L}^u_{(\ell_0,\mathbf{0})}(\mathcal{A}) = \mathcal{L}^u_{(\ell_0',\mathbf{0})}(\mathcal{A}')$ [BFH+01]. The intuition behind the construction is to store in the locations the information of which clock is above M and maintain those under $M+1$.

Bounded clocks are a powerful assumption that is usually required in related work, and will be necessary for most of our results. As the above transformation maintains untimed languages, this assumption comes without loss of generality for the emptiness and model-checking problems. From now on, we will assume that all clocks are bounded by some constant M, that all constants that appear in atomic constraints are included in $[-M, M]$, and we restrict $[\![\mathcal{A}]\!]$ to configurations in $L \times \mathbb{R}^{\mathcal{X}}_{\geqslant 0, <M}$, as executions that get out of this cube cannot continue and thus do not affect the languages of $\mathcal{A}$. Moreover, we change the universal guard $\top$ such that $[\![\top]\!]$ equals $\mathbb{R}^{\mathcal{X}}_{\geqslant 0, <M}$ instead of $\mathbb{R}^{\mathcal{X}}_{\geqslant 0}$.

2.3.2. Regions

We will rely on the crucial notion of regions, as introduced in the seminal work on timed automata [AD94]. Given a finite set of rational numbers $\mathcal{S} \subseteq \mathbb{Q}$, $\mathcal{S}$ is said to be of granularity $1/N$ if $\mathcal{S} \subseteq \mathbb{Q}_N$. Such N always exists, and one can find the smallest one by decomposing elements of $\mathcal{S}$ as irreducible fractions c/c' with $c \in \mathbb{Z}$, $c' \in \mathbb{N}_{>0}$ and use the least common multiple of all c' as N. A guard g is said to be of granularity $1/N$ if all constants in the atomic constraints of g form a set of granularity $1/N$. A zone is of granularity $1/N$ if it can be described by a guard of granularity $1/N$. Let $\mathsf{Guards}_N(\mathcal{X}, M)$ denote the set of guards over $\mathcal{X}$ bounded by M and of granularity $1/N$, and let $\mathsf{Guards}^{\mathrm{nd}}_N(\mathcal{X}, M)$ denote the non-diagonal ones. Given a finite set of guards $G \subseteq \mathsf{Guards}^{\mathrm{nd}}(\mathcal{X}, M)$, we can find N such that $G \subseteq \mathsf{Guards}^{\mathrm{nd}}_N(\mathcal{X}, M)$, by denoting $\mathcal{S} \subseteq \mathbb{Q}$ the set of constants used in atomic constraints of G and using N the smallest integer such that $\mathcal{S}$ is of granularity $1/N$. For all $a \in \mathbb{R}_{\geqslant 0}$, $\lfloor a \rfloor \in \mathbb{N}$ denotes the integral part of a, and $\mathsf{fract}(a) \in [0, 1)$ its fractional part, such that $a = \lfloor a \rfloor + \mathsf{fract}(a)$.

Definition 2.2. With respect to the set $\mathcal{X}$ of clocks, a granularity $N \in \mathbb{N}_{>0}$ and an upper bound $M \in \mathbb{N}_{>0}$, we define $1/N$-*regions* as subsets of valuations r characterised by a valuation $\iota \in \mathbb{R}^{\mathcal{X}}_{\geqslant 0, <M}$ called the integral part of r such that $\iota(\mathsf{x}) \in \mathbb{Q}_N$ for every $\mathsf{x} \in \mathcal{X}$, and an ordered partition $R_0 \uplus R_1 \uplus \cdots \uplus R_m$ splitting $\mathcal{X}$ into $m+1$ subsets. The ordered partition is denoted $0 = R_0 < R_1 < \cdots < R_m$, where R_0 can be empty but $R_i \neq \emptyset$ for $1 \leqslant i \leqslant m$.

A valuation ν in $\mathbb{R}^{\mathcal{X}}_{\geqslant 0, <M}$ belongs to r if

- for all $\mathsf{x} \in \mathcal{X}$, $\iota(\mathsf{x})N = \lfloor \nu(\mathsf{x})N \rfloor$;
- for all $\mathsf{x} \in R_0$, $\mathsf{fract}(\nu(\mathsf{x})N) = 0$;

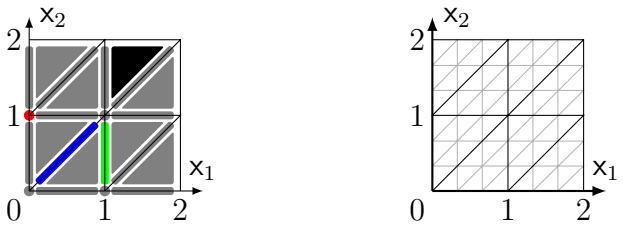


Figure 2.3.: All 1/1-regions in $\mathsf{Reg}_1(\{\mathsf{x}_1, \mathsf{x}_2\}, 2)$ on the left, their refinement of granularity 1/3 in $\mathsf{Reg}_3(\{\mathsf{x}_1, \mathsf{x}_2\}, 2)$ on the right.

- for all $0 \leqslant i \leqslant m$, for all $\mathsf{x}, \mathsf{y} \in R_i$, $\mathsf{fract}(\nu(\mathsf{x})N) = \mathsf{fract}(\nu(\mathsf{y})N)$.
- for all i, j such that $0 \leqslant i < j \leqslant m$, for all $\mathsf{x} \in R_i$ and all $\mathsf{y} \in R_j$, $\mathsf{fract}(\nu(\mathsf{x})N) < \mathsf{fract}(\nu(\mathsf{y})N)$.

With granularity $N = 1$, we recover the classical notion of regions from [AD94]. The set of valuations contained by a $1/N$-region r characterised by ι and $0 = \{\mathsf{x}_1^0, \ldots, \mathsf{x}_{m_0}^0\} < \{\mathsf{x}_1^1, \ldots, \mathsf{x}_{m_1}^1\} < \cdots < \{\mathsf{x}_1^m, \ldots, \mathsf{x}_{m_m}^m\}$ can be described by a formula, constructed as a conjunction of inequalities over $\mathcal{X}$: If we let $f(\mathsf{x})$ denote the term $(\mathsf{x} - \iota(\mathsf{x}))N$, and E^i denote the formula $f(\mathsf{x}_1^i) = \cdots = f(\mathsf{x}_{m_i}^i)$ for every $0 \leqslant i \leqslant m$, then $\nu \in r$ if and only if its coordinates satisfy

$$E^0 \wedge E^1 \wedge \cdots \wedge E^m \wedge 0 = f(\mathsf{x}_1^0) < f(\mathsf{x}_1^1) < \cdots < f(\mathsf{x}_1^m) < 1\,.$$

Therefore, every $1/N$-region is a zone of granularity $1/N$, associated to a guard in $\mathsf{Guards}_N(\mathcal{X}, M)$. We denote by $\mathsf{Reg}_N(\mathcal{X}, M)$ the set of $1/N$-regions bounded by M. It forms a finite partition of $\mathbb{R}^{\mathcal{X}}_{\geqslant 0, < M}$, and the number $|\mathsf{Reg}_N(\mathcal{X}, M)|$ of $1/N$-regions is polynomial in MN and exponential in $|\mathcal{X}|$. If ν is a valuation in $\mathbb{R}^{\mathcal{X}}_{\geqslant 0, < M}$, $[\nu]$ denotes the unique region that contains ν. Valuations in the same $1/N$-region satisfy the same guards in $\mathsf{Guards}_N(\mathcal{X}, M)$. In fact, zones associated to guards in $\mathsf{Guards}_N(\mathcal{X}, M)$ can be described as a finite union of regions in $\mathsf{Reg}_N(\mathcal{X}, M)$. If r is a $1/N$-region in $\mathsf{Reg}_N(\mathcal{X}, M)$, then the time-successor valuations in $\mathsf{PostTime}(r) \cap \mathbb{R}^{\mathcal{X}}_{\geqslant 0, < M}$ form a finite union of regions in $\mathsf{Reg}_N(\mathcal{X}, M)$, and the reset $r[\mathcal{Y} := 0]$ of $\mathcal{Y} \subseteq \mathcal{X}$ is a region in $\mathsf{Reg}_N(\mathcal{X}, M)$. A $1/N$-region r' is said to be a time successor of the $1/N$-region r if there exists $\nu \in r$, $\nu' \in r'$, and $d > 0$ such that $\nu' = \nu + d$.

If r is an $1/N$-region, let $\overline{r}$ denote its topological closure, *i.e.* the smallest zone that contains r associated to a non-strict guard.[6] The corners of r are valuations in $\overline{r}$ that belong to $\mathbb{Q}^{\mathcal{X}}_N$. If r is characterized by an integral part ι and a clock ordering $0 = R_0 < R_1 < \cdots < R_m$, then ι is a corner of r. If $m = 0$ then $r = \{\iota\}$, otherwise r does not include its corners but contains valuations arbitrarily close to them. The corners of r are the vertices of the polytope $\overline{r}$, such that $\overline{r}$ is their convex hull.

Example 2.3. Figure 2.3 represents the 24 regions of granularity $N = 1$ with upper bound $M = 2$ over two clocks. The green region is characterised by $\iota = (1, 0)$ and

[6] $\overline{r}$ is obtained by replacing every bound $(<, c)$ by $(\leqslant, c)$ in the DBM encoding r as a zone.

$0 = \{\mathsf{x}_1\} < \{\mathsf{x}_2\}$, corresponds to the formula $0 = \mathsf{x}_1 - 1 < \mathsf{x}_2 < 1$ and thus is equal to the zone $[\![\mathsf{x}_1 = 1 \wedge 0 < \mathsf{x}_2 < 1]\!]$. Its corners are the valuations $(1,0)$ and $(1,1)$. The red region is characterised by $\iota = (0,1)$ and $0 = \{\mathsf{x}_1, \mathsf{x}_2\}$. The black region is characterised by $\iota = (1,1)$ and $0 < \{\mathsf{x}_1\} < \{\mathsf{x}_2\}$. The blue region is characterised by $\iota = (0,0)$ and $0 < \{\mathsf{x}_1, \mathsf{x}_2\}$.

2.3.3. Region abstraction, region automaton

Definition 2.3. Given a timed automaton $\mathcal{A} = \langle L, \mathcal{X}, \Sigma, E\rangle$ such that all clocks are bounded by M and all guards belong to $\mathsf{Guards}^{\mathrm{nd}}_N(\mathcal{X}, M)$ for some granularity $1/N$, we define the *region abstraction* of $\mathcal{A}$ as the transition system $\langle L \times \mathsf{Reg}_N(\mathcal{X}, M), T\rangle$ labelled over $\mathsf{Reg}_N(\mathcal{X}, M) \times E$, where T contains all transitions $(\ell, r) \xrightarrow{r'', e} (\ell', r')$ such that $e = \ell \xrightarrow{a,g,\mathcal{Y}} \ell'$ is an edge of $\mathcal{A}$, r'' is a time-successor of r, $r'' \models g$ and $r''[\mathcal{Y} := 0] = r'$.

As there are finitely many regions, the region abstraction of $\mathcal{A}$ is a finite transition system, where paths p represent a sequence of regions alternating between letting time elapse and taking edges, following some path π in $\mathcal{A}$. We say that π contains the region path p, and when π is clear from context we sometimes denote $r_1 \xrightarrow{\text{delay}} r_2 \xrightarrow{a_1,g_1,\mathcal{Y}_1} r_3 \dots$ the path $\mathsf{p} = (\ell_1, r_1) \xrightarrow{r_2,e_1} (\ell_2, r_3) \dots$ with $e_1 = (\ell_1, a_1, g_1, \mathcal{Y}_1, \ell_2)$. The states of the region abstraction are called region states, and its paths are called region paths. From an execution in $\mathcal{A}$ described by $\rho = (\ell_0, \nu_0) \xrightarrow{d_1,e_1} (\ell_1, \nu_1) \xrightarrow{d_2,e_2} \dots$, we can construct a region path $\mathsf{p} = (\ell_0, [\nu_0]) \xrightarrow{[\nu_0+d_1],e_1} (\ell_1, [\nu_1]) \xrightarrow{[\nu_1+d_2],e_2} \dots$, and say that ρ follows p.

From the region abstraction $(L \times \mathsf{Reg}_N(\mathcal{X}, M), T)$, we can construct a timed automaton $\mathcal{R}_N(\mathcal{A}) = \langle L \times \mathsf{Reg}_N(\mathcal{X}, M), \mathcal{X}, \Sigma, E'\rangle$, called the *region automaton* of $\mathcal{A}$, whose locations are region states and such that E' is defined by transforming every transition $(\ell, r) \xrightarrow{r'', e} (\ell', r')$ in T, where e is labelled by $(a, g, \mathcal{Y})$, into an edge $(\ell, r) \xrightarrow{a,g'',\mathcal{Y}} (\ell', r')$, with $[\![g'']\!] = r'' \subseteq [\![g]\!]$. Every execution in $\mathcal{A}$ exists in $\mathcal{R}_N(\mathcal{A})$ as an execution following a region path p, and conversely every execution in $\mathcal{R}_N(\mathcal{A})$ following some region path p is contained in the path π of $\mathcal{A}$ followed by p. Then, $\mathcal{A}$ and $\mathcal{R}_N(\mathcal{A})$ contain the same executions, therefore they have the same timed and untimed languages and are equivalent w.r.t. the emptiness and model-checking problems.

Proposition 2.1 ([AD94])**.** *Consider a region path* p *starting in* (ℓ, r)*. For all valuations* ν *in* r*, there exists in* $\mathcal{R}_N(\mathcal{A})$ *an execution following* p *and starting from* (ℓ, ν)*. If* p *is finite and ends in* (ℓ', r')*, there also exists in* $\mathcal{R}_N(\mathcal{A})$ *an execution following* p *and ending at* (ℓ', ν') *for every valuation* $\nu' \in r'$*.*

One can then deduce that for every initial configuration (ℓ_0, ν_0), the language $\mathcal{L}_{(\ell_0,[\nu_0])}$ of the region abstraction, defined as the infinite words over Σ induced by region paths starting from $(\ell_0, [\nu_0])$, is equal to the untimed language $\mathcal{L}^u_{(\ell_0,\nu_0)}(\mathcal{R}_N(\mathcal{A}))$ [AD94]. It follows that $\mathcal{A}$ satisfies the emptiness (resp. model-checking) problem with initial configuration (ℓ_0, ν_0) and ω-regular objective $\mathcal{L}^u_{\mathsf{t}}$ if and only if the region abstraction, seen as a transition system labelled over actions in Σ, satisfies the emptiness (resp. model-checking) problem

with initial state $(\ell_0, [\nu_0])$ and objective $\mathcal{L}^u_t$. As the region abstraction is a finite transition system, those problems can be solved using the region abstraction of $\mathcal{A}$. For reachability or Büchi objectives, the time-complexity of this procedure becomes polynomial in the size of the region abstraction of $\mathcal{A}$, therefore exponential in the size of $\mathcal{A}$.

By using the region abstraction, the emptiness problem can be shown to be PSPACE-complete for reachability or Büchi conditions [AD94].

2.3.4. Integer constants

A timed automaton $\mathcal{A}$ can be scaled by $N \in \mathbb{N}_{>0}$ by multiplying every constant in the guards of $\mathcal{A}$ by N. The executions in the scaled automaton are scaled versions of the executions in $\mathcal{A}$, where delays and valuations are multiplied by N. This means that we can restrict the constants to integer values without loss of generality, as we only care about the properties of the untimed words induced by those executions. Formally, let $\mathcal{A}$ be a timed automaton, with guards in $\mathsf{Guards}^{\mathrm{nd}}_N(\mathcal{X}, M)$, such that $N \in \mathbb{N}_{>0}$ is as small as possible for the constants in $\mathcal{A}$. There exists a timed automaton $\mathcal{A}'$ with guards in $\mathsf{Guards}^{\mathrm{nd}}_1(\mathcal{X}, MN)$ such that $\mathcal{A}$ and $\mathcal{A}'$ have the same untimed language. Moreover, if constants are encoded in binary, the size of $\mathcal{A}'$ is at most quadratic in the size of $\mathcal{A}$ [AD94].

From now on, we assume that the guards in the timed automata we consider have granularity $1/N = 1$, such that every constant that appears in atomic constraints belong to $\mathbb{Z}$. In this case, we omit N from previous notations about regions, such that $1/N$-regions are simply called regions, $\mathsf{Reg}_N(\mathcal{X}, M)$ is denoted $\mathsf{Reg}(\mathcal{X}, M)$, and $\mathcal{R}_N(\mathcal{A})$ becomes $\mathcal{R}(\mathcal{A})$.

2.3.5. Zone abstraction, symbolic algorithms

While the region abstraction gives optimal complexity results for reachability and Büchi conditions, techniques based on constructing the region abstraction struggle with run-time efficiency, mostly because of the large state-space to explore. Practical implementations rely on so-called *symbolic* techniques instead, where zones are used to represent sets of regions and where the region abstraction is constructed on the fly.

Let $\mathsf{Zones}_N(\mathcal{X}, M)$ be the set of zones associated with guards in $\mathsf{Guards}_N(\mathcal{X}, M)$. This is a finite set since these zones are finite unions of regions in $\mathsf{Reg}_N(\mathcal{X}, M)$. Let $\mathsf{Zones}(\mathcal{X}, M)$ denote $\mathsf{Zones}_1(\mathcal{X}, M)$. The following elementary operations can be defined on zones $Z \in \mathsf{Zones}(\mathcal{X}, M)$:

- Time elapse, as $\mathsf{PostTime}_{<M}(Z) = \mathsf{PostTime}(Z) \cap \mathbb{R}^{\mathcal{X}}_{\geqslant 0, <M}$.
- Intersection with a guard $g \in \mathsf{Guards}^{\mathrm{nd}}_1(\mathcal{X}, M)$, as $Z \cap [\![g]\!]$.
- Reset of $\mathcal{Y} \subseteq \mathcal{X}$, with $\mathsf{Reset}_{\mathcal{Y}}(Z) = Z[\mathcal{Y} := 0]$.

Their output is also a zone in $\mathsf{Zones}(\mathcal{X}, M)$, which let us define the successor of a zone Z by an edge $e = (\ell, a, g, \mathcal{Y}, \ell')$ as $\mathsf{Post}_e(Z) = \mathsf{Reset}_{\mathcal{Y}}([\![g]\!] \cap \mathsf{PostTime}_{<M}(Z))$.

Given a timed automaton $\mathcal{A} = \langle L, \mathcal{X}, \Sigma, E \rangle$ such that all clocks are bounded by M and all guards belong to $\mathsf{Guards}_1^{\mathrm{nd}}(\mathcal{X}, M)$, we define the *zone abstraction*[7] of $\mathcal{A}$ as the transition system $\langle L \times \mathsf{Zones}(\mathcal{X}, M), T \rangle$ labelled over E, where states (ℓ, Z) are called zone states and where T contains all transitions $(\ell, Z) \xrightarrow{e} (\ell', Z')$ such that e is an edge of $\mathcal{A}$ from ℓ to ℓ', $Z' = \mathsf{Post}_e(Z)$ and $Z' \neq \emptyset$. The zone abstraction of $\mathcal{A}$ is a finite transition system, and can be seen as a symbolic representation of the region abstraction of $\mathcal{A}$ by interpreting zones as sets of regions. Recall that if ν is a valuation then the region $[\nu]$ is a zone that contains ν. The zone abstraction inherits properties from the region abstraction, most notably its language over Σ starting from $(\ell_0, [\nu_0])$ is equal to the untimed language of $\mathcal{A}$ starting from (ℓ_0, ν_0), and therefore the zone abstraction is equivalent to $\mathcal{A}$ w.r.t. the emptiness and model-checking problems.

Given an initial configuration (ℓ_0, ν_0), we say that a zone state (ℓ, Z) is reachable from (ℓ_0, ν_0) if there exists a path in the zone abstraction from $(\ell_0, [\nu_0])$ to (ℓ, Z). The zone abstraction contains a number of states exponential in the size of $\mathcal{A}$, but the ones not reachable from (ℓ_0, ν_0) can be ignored. Algorithm 2.1 describes the classical forward exploration technique, computing the set of all zone states reachable from an initial configuration (ℓ_0, ν_0). It uses two sets of zone states, Passed and Waiting, to explore the zone abstraction, starting from $(\ell_0, Z_0) = (\ell_0, [\nu_0])$. At every step, we select a zone state (ℓ, Z) of Waiting, check if it is in Passed, and if not we add (ℓ, Z) to Passed and its successors to Waiting. All elementary operations on zones can be implemented efficiently using DBMs in normal form (see Section 2.2).

Algorithm 2.1: Forward exploration of the zone abstraction

Input : A timed automaton $\langle L, \mathcal{X}, \Sigma, E \rangle$, an initial zone state (ℓ_0, Z_0)
Output : All zone states reachable from (ℓ_0, Z_0)

```
1 Passed ← ∅
2 Waiting ← {(ℓ0, Z0)}
3 while Waiting ≠ ∅ do
4 |   select and remove a zone state (ℓ, Z) from Waiting
5 |   if for all (ℓp, Zp) ∈ Passed, ℓ ≠ ℓp or Z ≠ Zp then
6 |   |   add (ℓ, Z) to Passed
7 |   |   for e = (ℓ, a, g, 𝒴, ℓ') ∈ E do
8 |   |   |   if Post_e(Z) ≠ ∅ then add (ℓ', Post_e(Z)) to Waiting
9 return Passed
```

Let us now focus on the existence problem with a reachability condition, where we are given a set of target locations L_{t} and want to decide if they can be reached by an execution starting from (ℓ_0, ν_0). This problem can be solved by applying any algorithm dealing with reachability objectives on finite transition systems on the zone abstraction. For example, one could use Algorithm 2.1, and test if some reachable state (ℓ, Z) satisfies

[7] usually called zone graph

$\ell \in L_{\mathsf{t}}$. However, this algorithm can be improved, by testing for zone inclusion instead of equality in line 5 ($Z \neq Z_p$ becomes $Z \not\subseteq Z_p$). This is correct because if $\ell = \ell_p$ and $Z \subseteq Z_p$ then all locations reachable from configurations in (ℓ, Z) are also reachable from configurations in (ℓ_p, Z_p), therefore exploring zone states reachable from (ℓ, Z) is not needed for reachability conditions. In this case, Algorithm 2.1 computes a subset of the reachable zone states that covers reachable configurations, *i.e.* there exists an execution from (ℓ_0, ν_0) to (ℓ, ν) if and only if there exists $(\ell, Z) \in \mathsf{Passed}$ such that $\nu \in Z$. One can further reduce the explored state-space by using finer approaches, called extrapolation techniques [Bou03, BBFL03, HKSW11, BBLP04, HSW11], where the zone Z added to Passed in line 6 is enlarged into some $\alpha(Z) \supseteq Z$ to speed up the computation. The termination of Algorithm 2.1 relies on the finite number of zones in $\mathsf{Zones}(\mathcal{X}, M)$, and therefore on the bounded clocks assumption. This assumption can be removed using extrapolation techniques where operator α has finite image.

Part II.

Robust controller synthesis

Introduction

In this part we study problems related to the *robustness* of controllers. As mentioned in the general introduction, the semantics of timed automata is a mathematical idealisation, where delays are chosen with an infinite precision. Therefore, alternative semantics have been proposed in order to ensure that a property satisfied by a timed automaton $\mathcal{A}$ carries over to implementations modelled by $\mathcal{A}$. Integrating the robustness question in the verification of real-time systems has attracted attention in the community, and the recent works include, for instance, robust model checking for timed automata under clock drifts [RPV17], Lipschitz robustness notions for timed systems [HOS16], quantitative robust synthesis for timed automata [BBF+18]. Stability analysis and synthesis of stabilizing controllers in hybrid systems are a closely related topic, see *e.g.* [PS16, PGS17].

For the model-checking problem that asks if every execution in $\mathcal{A}$ satisfies some objective, an approach consists in *enlarging* the guards by a small amount $\delta > 0$, *e.g.* so that $2 \leqslant \mathsf{x} < 3$ becomes $2 - \delta \leqslant \mathsf{x} < 3 + \delta$, thus defining an enlarged timed automaton $\mathcal{A}_\delta$. Indeed, if $\mathcal{A}_\delta$ satisfies the model-checking problem, so does $\mathcal{A}$, and there is always some margin δ left around executions of $\mathcal{A}$, ensuring that the specification is ensured in a robust manner. The qualitative robust model-checking problem then asks if there exists some $\delta > 0$ such that $\mathcal{A}_\delta$ satisfies the model-checking problem. It was shown in [DWDMR08, Pur00] to be PSPACE-complete for safety properties and LTL formulæ [BMR06]. A quantitative variant, asking what is the greatest $\delta \geqslant 0$ such that $\mathcal{A}_\delta$ satisfies the model-checking problem, was also studied in [JR11].

A symmetrical approach can be followed for the emptiness problem, that asks if there exists an execution in $\mathcal{A}$ satisfying the specification. In this context, the goal is to restrict the semantics, so as to remove executions that would not be implementable in a real-world situation (were infinitesimal imprecisions on measurements are bound to appear). In this case, guards are *shrunk*, so that $2 \leqslant \mathsf{x} < 3$ becomes $2 + \delta \leqslant \mathsf{x} < 3 - \delta$. If the shrunk system satisfies the emptiness problem, so does the original timed automaton, and every guard is satisfied with some margin for error. The intuition is that the emptiness problem searches for an execution, modelling an implementation, that satisfies the specification.

However, we argue that this shrinking approach does not provide a satisfying notion of robustness for infinite behaviours, as it does not let imprecisions accumulate, and does not prevent Zeno behaviours:

Example 1. Consider a simple timed automaton, with a single location ℓ_0, and an edge $\ell_0 \xrightarrow{\mathsf{x}<1,\emptyset} \ell_0$. This edge can be followed infinitely many times, using increasingly small delays. In contrast, a real implementation must run on a processor with a fixed frequency, and thus cannot allow any infinite execution. Shrinking the guard $\mathsf{x} < 1$ by $\delta \in (0, 1/2)$ results in guard $\delta \leqslant \mathsf{x} < 1 - \delta$. The shrunk edge can still be followed infinitely many

times, by aiming for a convergence point in $(\delta, 1-\delta) \neq \emptyset$.

We study the emptiness problem with a Büchi acceptance condition: it consists in determining whether there exists an accepting infinite execution. As such, we will use another notion of robustness, based on perturbations. The situation is modelled as a two-player game $\mathcal{G}_\delta$, between the controller that chooses edges and delays, and the antagonistic environment that can perturb each delay using a value chosen in the interval $[-\delta, \delta]$. This approach forbids executions that rely on Zeno behaviours, or even non-Zeno behaviours requiring infinite precision [CHR02]. In contrast with the shrinking semantics, controller can react to the perturbations, and compensate for them by changing future delays.

Example 2. Coming back to Example 1, in the perturbation semantics the environment can ask every time elapse to be at least $\delta > 0$, and as such after $1/\delta$ steps the value of x is greater than 1, and infinite executions are no longer possible.

For a fixed value of $\delta > 0$, the *robust controller synthesis problem* asks if controller has a winning strategy in $\mathcal{G}_\delta$. Solving it is useful if we know the characteristics of the hardware on which the controller will be implemented. This problem has been shown to be decidable in [CHP11], and also for a related model in [LLTW14]. These algorithms are based on using $1/N$-regions, with $N \in \mathbb{N}_{>0}$ a granularity such that $\delta \in \mathbb{Q}_N$. The perturbation δ may be small compared to the constants appearing in the guards of the timed automaton, and thus we will need a large N. As the overall complexity of these approaches is sensitive to region-granularity (the state-space is linear in $N^{|\mathcal{X}|}$), they may not yield practical algorithms.

In the context of synthesis, one may not know in advance what is the amplitude on perturbations δ that the controller will face. In this case, a preliminary step consists in studying the *qualitative* robust controller synthesis problem, that asks if there exists $\delta > 0$ such that the controller has a winning strategy in $\mathcal{G}_\delta$. If the answer is yes, one would ideally obtain a controller and a value for $\delta > 0$, that can then be used to guide hardware choices.

The qualitative problem enjoys promising complexity results: it has been proven in [SBMR13] that it is PSPACE-complete, thus no harder than the exact setting with $\delta = 0$ [AD94], and it does not require the use of regions of lower granularity. However, the algorithm of [SBMR13] heavily relies on regions, and more precisely on an abstraction that refines the one of regions, namely folded orbit graphs. Hence, it is not amenable to implementation.

Our objective is to provide an efficient symbolic algorithm for solving the qualitative problem. To this end, we target the use of *zones* instead of regions, as they allow an on-demand partitioning of the state space. Moreover, the algorithm we develop explores the reachable state-space in a *forward* manner. This is known to lead to better performances, as witnessed by the successful tool Uppaal Tiga based on forward algorithms for solving controller synthesis problems [CDF+05].

Our algorithm can be understood as an adaptation to the timed setting of a classical nested breadth-first search algorithm for transition systems with Büchi objectives. This

algorithm is searching for a winning lasso, *i.e.* a path from the initial location to a target, followed by a cycle around the target. In the timed setting, this search takes place in the zone abstraction. A similar approach is detailed in [LOD+13], where a major difficulty consists in checking if a winning cycle can be iterated for an infinite amount of steps. Indeed, it is explained that standard speedup techniques using zone inclusions to reduce the search-space may not preserve Büchi emptiness.

In the context of robustness, we need to additionally preserve the robust iterability of winning cycles, which is not maintained by the techniques of [LOD+13]. The key argument of [SBMR13] was the notion of aperiodic folded orbit graph of a path in the region automaton, thus tightly connected to regions. Lifting this notion to zones seems impossible as it makes an important use of the fact that valuations in regions are time-abstract bisimilar, which is not the case for zones. We will address this difficulty by computing the reachability relations of zone paths, and showing that inclusion of reachability relations is a complete termination criterion.

On positive instances, our algorithm outputs a winning strategy for controller, in the form of a lasso in the zone abstraction. At this point, we know that there exists a value of $\delta > 0$ that controller wins against, but we are not provided with such a δ. We overcome this issue by studying the more ambitious *quantitative* robust controller synthesis problem, that asks what is the greatest $\delta > 0$ such that the controller has a winning strategy in $\mathcal{G}_\delta$. The decidability of this problem is unknown, but we solve it on the particular case where controller follows a lasso of the zone graph, and are thus able to extract not only a valid δ, but the greatest one.

We introduce our notion of robustness to perturbations in Chapter 3, detail the result of [SBMR13] in Chapter 4, present our symbolic procedure in Chapter 5, and study the qualitative problem in Chapter 6.

3. The perturbation game

Consider a timed automaton $\mathcal{A} = \langle L, \mathcal{X}, \Sigma, E \rangle$, equipped with an initial location ℓ_0 and a Büchi condition $L_\mathsf{t} \subseteq L$ as a set of target locations. We are interested in the locations reached by executions of $\mathcal{A}$, but not by the actions performed by those executions. Therefore, we assume that every edge has a unique action in Σ associated to it, and remove actions from our notations, such that $\mathcal{A}$ is written $\langle L, \mathcal{X}, E \rangle$ and an edge $\ell \xrightarrow{a,g,\mathcal{Y}} \ell'$ is denoted $\ell \xrightarrow{g,\mathcal{Y}} \ell'$ instead. We assume that clocks $\mathcal{X}$ in $\mathcal{A}$ are bounded by $M \in \mathbb{N}_{>0}$ and that every constant in the guards is an integer, such that every guard g in $\mathcal{A}$ belongs to $\mathsf{Guards}_1^{\mathrm{nd}}(\mathcal{X}, M)$. With robustness in mind, we argue that null delays should not be authorized, and restrict the transition system $[\![\mathcal{A}]\!]$ to transitions $(\ell, \nu) \xrightarrow{d,e} (\ell', \nu')$ with delay $d > 0$. Previous results still hold in this case, as operators and definitions can be adjusted accordingly. For example, operator $\mathsf{PostTime}$ will be denoted $\mathsf{PostTime}_{>0}$ in this case, and we can use it to define adapted variants of the Post operator and of the forward exploration algorithm over zones.

We study the robustness problem introduced in [SBMR13], that is stated in terms of games where a controller fights against an environment. After a prefix of an execution, the controller will have the capability to choose delays and edges to fire, whereas the environment perturbs the delays chosen by the controller with a small parameter $\delta \geqslant 0$. The aim of the controller will be to find a strategy so that, no matter how the environment plays, he is ensured to generate an infinite execution satisfying a Büchi condition.

Definition 3.1. Given a timed automaton $\mathcal{A} = \langle L, \mathcal{X}, E \rangle$ of initial location ℓ_0 and Büchi condition L_t, and a maximal perturbation $\delta \in \mathbb{Q}_{\geqslant 0}$, the *perturbation game* is a two-player turn-based game $\mathcal{G}_\delta(\mathcal{A})$ between a controller and an environment. Its state space is partitioned into $S_{\mathsf{Ctrl}} \uplus S_{\mathsf{Env}}$ where $S_{\mathsf{Ctrl}} = L \times \mathbb{R}^{\mathcal{X}}_{\geqslant 0, <M}$ belongs to the controller, and $S_{\mathsf{Env}} = L \times \mathbb{R}^{\mathcal{X}}_{\geqslant 0, <M} \times \mathbb{R}_{>0} \times E$ to the environment. The initial state is $(\ell_0, \mathbf{0}) \in S_{\mathsf{Ctrl}}$. From each state $(\ell, \nu) \in S_{\mathsf{Ctrl}}$, there is a transition to $(\ell, \nu, d, e) \in S_{\mathsf{Env}}$ with $e = (\ell, g, \mathcal{Y}, \ell') \in E$ whenever $d > \delta$, and $\nu + d + \varepsilon \models g$ for all $\varepsilon \in [-\delta, \delta]$. Then, from each state $(\ell, \nu, d, (\ell, g, \mathcal{Y}, \ell')) \in S_{\mathsf{Env}}$, there is a transition to $(\ell', (\nu + d + \varepsilon)[\mathcal{Y} := 0]) \in S_{\mathsf{Ctrl}}$ for all $\varepsilon \in [-\delta, \delta]$.

A play of $\mathcal{G}_\delta(\mathcal{A})$ is a finite or infinite path $q_0 \xrightarrow{t_1} q_1 \xrightarrow{t_2} q_2 \cdots$ where $q_0 = (\ell_0, \mathbf{0})$ and t_i is a transition from state q_{i-1} to q_i for all $i > 0$. It is said to be maximal if it is infinite or cannot be extended with any transition. A strategy for the controller is a function σ_{Ctrl} mapping each non-maximal play ending in some $(\ell, \nu) \in S_{\mathsf{Ctrl}}$ to a pair (d, e) where $d > 0$ and $e \in E$ such that there is a transition from (ℓ, ν) to (ℓ, ν, d, e). A strategy for the environment is a function σ_{Env} mapping each finite play ending in (ℓ, ν, d, e) to a state (ℓ', ν') related by a transition. A play gives rise to a unique execution

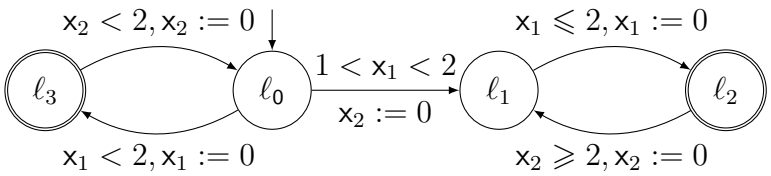


Figure 3.1.: A timed automaton, with initial location ℓ_0 and targets $\{\ell_2, \ell_3\}$.

of $[\![\mathcal{A}]\!]$ by only keeping the states in S_{Ctrl}. For a pair of strategies $(\sigma_{\mathsf{Ctrl}}, \sigma_{\mathsf{Env}})$, we let $\mathsf{play}^{\delta}_{\mathcal{A}}((\ell_0, \mathbf{0}), \sigma_{\mathsf{Ctrl}}, \sigma_{\mathsf{Env}})$ denote the execution associated with the unique maximal play of $\mathcal{G}_\delta(\mathcal{A})$ that follows the strategies. Controller's strategy σ_{Ctrl} is winning (with respect to the Büchi condition L_t) if for all strategies σ_{Env} of the environment, $\mathsf{play}^{\delta}_{\mathcal{A}}((\ell_0, \mathbf{0}), \sigma_{\mathsf{Ctrl}}, \sigma_{\mathsf{Env}})$ is infinite and visits infinitely often some location of L_t.

If $\delta = 0$, then the environment has only a single choice of $\varepsilon \in [-\delta, \delta]$, so that deciding if controller has a winning strategy is equivalent to solving the emptiness problem with a Büchi condition, which is PSPACE-complete [AD94]. We define three different problems, depending on whether the maximal perturbation δ is known or not.

Definition 3.2. The *robust controller synthesis problem* asks, given a timed automaton $\mathcal{A}$ and a maximal perturbation $\delta \geqslant 0$, whether the controller has a winning strategy in $\mathcal{G}_\delta(\mathcal{A})$.

Definition 3.3. The *qualitative robust controller synthesis problem* asks, given a timed automaton $\mathcal{A}$, whether there exists $\delta > 0$ such that the controller has a winning strategy in $\mathcal{G}_\delta(\mathcal{A})$.

Definition 3.4. The *quantitative robust controller synthesis problem* is a computation problem and asks, given a timed automaton $\mathcal{A}$, what is the supremum of $\delta \geqslant 0$ such that the controller has a winning strategy in $\mathcal{G}_\delta(\mathcal{A})$.

Those problems can also be expressed with a reachability condition instead of a Büchi condition: controller wins $\mathcal{G}_\delta(\mathcal{A})$ if he can ensure visiting some location of L_t at least once. Büchi conditions are more challenging, as we need to deal with the issues of robustly reaching a target, but also with the robustness of infinite paths along a cycle, where perturbations may accumulate.

Example 3.1. The controller has a winning strategy in $\mathcal{G}_\delta(\mathcal{A})$, with $\mathcal{A}$ the automaton of Figure 3.1, for all possible values of $\delta < 1/2$. Indeed, he can follow the cycle $\ell_0 \to \ell_3 \to \ell_0$ by always picking time delay $1/2$ so that, when arriving in ℓ_3 (resp. ℓ_0) after the perturbation of the environment, clock x_2 (resp. x_1) has a valuation in $[1/2 - \delta, 1/2 + \delta]$. Therefore, he can play forever following this positional strategy. For $\delta \geq 1/2$, the environment can enforce reaching ℓ_3 with a value for x_2 at least equal to 1. The guard $x_2 < 2$ of the next edge to ℓ_0 cannot be guaranteed, and therefore the controller cannot win $\mathcal{G}_\delta(\mathcal{A})$. In [SBMR13], it is shown that the cycle around ℓ_2 does not provide a winning strategy for the controller for any value of $\delta > 0$ since perturbations accumulate so that the controller can only play it a finite number of times in the worst case.

We are interested in cases where the maximal perturbation is not known, and it is simply assumed that some infinitesimal perturbations will happen in real-life executions of the system. Therefore, our results mostly focus on deciding the qualitative problem. When it is satisfied, we can provide a strategy for controller that resists some perturbation, and we can solve the quantitative problem on the automaton restricted to that strategy, *i.e.* we can compute the greatest perturbation admissible for this controller's strategy.

4. A region-based approach

By [SBMR13], the qualitative robust controller synthesis problem is known to be PSPACE-complete. Their solution is based on the region automaton of $\mathcal{A}$. We are seeking for a more practical solution using zones. In this chapter, we introduce some of the machinery used to obtain the results of [SBMR13], and extend one of their results to consider paths in $\mathcal{A}$ instead of region paths. We will use this generalisation in the next chapter.

A classical way to approach Büchi conditions in transition systems is to look for lassos $\pi_0\pi^\omega$, where π_0 is a path from the initial state to a target state, and where π is a cycle around the target state. The techniques of [SBMR13] look for such lassos $\mathsf{p}_0\mathsf{p}^\omega$ in the region abstraction, and then test if they can be robustly followed. We will start by explaining how to check if a finite region path p_0 can be robustly followed, and then focus on checking if a region cycle p can be robustly iterated infinitely many times.

4.1. Robustness of region paths

Consider a finite region path p. We formalise the notion of following p in a manner that resists perturbations by taking a backward analysis point of view.

4.1.1. Controllable predecessors

We define the following operators over sets of valuations Z, symmetrically to the ones used for symbolic forward analysis algorithms over the zone abstraction: $\mathsf{PreTime}_{>t}(Z)$ is the set of valuations such that a time delay of more than t time units leads to Z, $\mathsf{Unreset}_{\mathcal{Y}}(Z)$ is the set of valuations in $\mathbb{R}^{\mathcal{X}}_{\geqslant 0,<M}$ that end in Z when the clocks in $\mathcal{Y}$ are reset.

Definition 4.1. Consider an edge $e = (\ell, g, \mathcal{Y}, \ell')$. For every set $Z \subseteq \mathbb{R}^{\mathcal{X}}_{\geqslant 0,<M}$, we define the *predecessors of Z by e* as $\mathsf{Pre}_e(Z) = \mathsf{PreTime}_{>0}(\llbracket g \rrbracket \cap \mathsf{Unreset}_{\mathcal{Y}}(Z))$.

We extend this operator to a path π by induction, such that if π is of length 0, $\mathsf{Pre}_\pi(Z) = Z$, and if $\pi = \pi' e$ with e an edge, $\mathsf{Pre}_\pi(Z) = \mathsf{Pre}_{\pi'}(\mathsf{Pre}_e(Z))$.

From a robustness perspective, we also consider the operator $\mathsf{shrink}_{[-\delta,\delta]}(Z)$ defined as the set of valuations ν such that $\nu + [-\delta, \delta] \subseteq Z$ introduced in [SBM11]. Given an edge $e = (\ell, g, \mathcal{Y}, \ell')$, a set $Z \subseteq \mathbb{R}^{\mathcal{X}}_{\geqslant 0,<M}$ and a fixed $\delta \geqslant 0$, we define the *controllable predecessors of Z* as follows:

$$\mathsf{CPre}^{\delta}_e(Z) = \mathsf{PreTime}_{>\delta}(\mathsf{shrink}_{[-\delta,\delta]}(\llbracket g \rrbracket \cap \mathsf{Unreset}_{\mathcal{Y}}(Z)))\,.$$

Intuitively, $\mathsf{CPre}^{\delta}_{e}(Z)$ is the set of valuations from which the controller can ensure reaching Z in one step, following the edge e, no matter of the perturbations of amplitude at most δ of the environment. In fact, it can delay in $\mathsf{shrink}_{[-\delta,\delta]}([\![g]\!] \cap \mathsf{Unreset}_{\mathcal{Y}}(Z))$ with a delay of at least δ, where under every perturbation in $[-\delta, \delta]$, the valuation satisfies the guard, and it ends, after reset, in Z. We extend this operator to a path π by induction, denoting it by $\mathsf{CPre}^{\delta}_{\pi}$: if π is of length 0, $\mathsf{CPre}^{\delta}_{\pi}(Z) = Z$, and if $\pi = \pi' e$ with e an edge, $\mathsf{CPre}_{\pi}(Z) = \mathsf{CPre}_{\pi'}(\mathsf{CPre}_{e}(Z))$. Note that $\mathsf{CPre}^{0}_{\pi} = \mathsf{Pre}_{\pi}$ is the usual predecessor operator without perturbation. We can also define those operators on region paths instead of paths by applying them to the region automaton.

If we consider a fixed $\delta \in \mathbb{Q}_{\geqslant 0}$, and find a granularity $N \in \mathbb{N}_{>0}$ such that $\delta N \in \mathbb{N}$, then all of those operations are closed over $\mathsf{Zones}_N(\mathcal{X}, M)$, and given a DBM describing Z they can be effectively computed in time cubic in $|\mathcal{X}|$ [SBM11]. Now, consider a region path p from (ℓ, r) to (ℓ', r'). If controller starts in (ℓ, r) at some valuation $\nu \in r$ and wants to reach (ℓ', r') in $\mathcal{G}_{\delta}(\mathcal{A})$ by following p, he has a strategy ensuring it if and only if $\nu \in \mathsf{CPre}^{\delta}_{\mathsf{p}}(r')$.

For the qualitative problem, we will perform computations that are parametrised by δ.

4.1.2. Shrunk DBMs

Definition 4.2. We say that p from (ℓ, r) to (ℓ', r') can be robustly followed if $r \cap \mathsf{CPre}^{\delta}_{\mathsf{p}}(r') \neq \emptyset$ for some $\delta > 0$.

We will use a parametrised extension of DBMs, namely *shrunk DBMs*, that were introduced in [SBM11] in order to study the parametrised state space of timed automata. Intuitively, our goal is to express *shrinkings* of guards, *e.g.* sets of states satisfying constraints of the form $g = 1 + \delta < \mathsf{x} < 2 - \delta \wedge 2\delta < \mathsf{y}$, where δ is a parameter to be chosen.

Formally, a *shrunk bound* is a pair $((\prec, c), p)$ with $(\prec, c)$ a member of the semiring of bounds $(\mathsf{Bounds}(\mathbb{Z}), \min, +)$ such that $\prec \in \{\leqslant, <\}$ and $c \in \mathbb{Z}$, and p a non-negative integer in $\mathbb{N} \cup \{\infty\}$, that represents the bound $\prec c - \delta p$ for small enough values of $\delta \geqslant 0$: A real a satisfies $((\prec, c), p)$ for $\delta = 0$ if $a \prec c$, it satisfies $((\prec, c), p)$ for $\delta > 0$ if $a \prec c - \delta p$ holds for some $\delta > 0$ small enough. The shrunk bound $((\prec, c), \infty)$ can be satisfied for $\delta = 0$, but is never satisfied for $\delta > 0$. This will appear when perturbations can accumulate. We also define the shrunk bounds $((<, -\infty), \infty)$ and $((<, +\infty), 0)$, representing constraints that are respectively never and always satisfied, for both $\delta = 0$ and $\delta > 0$.

If m, m' are bounds in $\mathsf{Bounds}(\mathbb{Z})$, recall that we write $m \preccurlyeq m'$ if m is at least as strong as m' (*i.e.* $m = \min(m, m')$), and $m \prec m'$ if $m \preccurlyeq m' \wedge m \neq m'$. We say that a shrunk bound $((\prec, c), p)$ is at least as strong as $((\prec', c'), p')$, if either $p, p' \in \mathbb{N}$ and $(\prec, c) \prec (\prec', c') \vee ((\prec, c) = (\prec', c') \wedge p \geqslant p')$, or $p = \infty \wedge p' \in \mathbb{N}$, or $p = p' = \infty \wedge m \preccurlyeq m'$. This forms a total order over shrunk bounds, where the strongest bound is $((<, -\infty), \infty)$ and the weakest bound is $((<, +\infty), 0)$. If $((\prec, c), p)$ is at least as strong as $((\prec', c'), p')$ then for all $\delta > 0$ and all $a \in \mathbb{R}$, $a \prec c - \delta p$ implies $a \prec' c' - \delta p'$.

We define operations min and inf over shrunk bounds in an intuitive way (min returns

the strongest bound and inf the weakest bound at least as strong as its inputs), and an addition $(m,p)+(m',p')$: it equals $((<,+\infty),0)$ if $m+m'=(<,+\infty)$, $((<,-\infty),\infty)$ if $m+m'=(<,-\infty)$, and $(m+m',p+p')$ otherwise, with $m+m'$ the addition of bounds and $p+p'$ the standard addition over $\mathbb{N}$ extended with $c+\infty=\infty$ for all $c\in\mathbb{N}$ and $\infty+\infty=\infty$. This defines a semiring of operators min and $+$, that we call the tropical semiring of shrunk bounds. It is an ordered semiring since min is selective, and it has closure because it forms a complete join semilattice with the order induced by min, where join is the inf operator. The closure $(m,p)^{(*)}$ of a shrunk bound (m,p) with $p\in\mathbb{N}_{>0}$ is $(m^{(*)},\infty)$ if $m\preccurlyeq(\leqslant,0)$, $(m^{(*)},0)$ otherwise. The closure of $(m,0)$ is $(m^{(*)},0)$, and the closure of (m,∞) is $(m^{(*)},\infty)$. This means that we can construct DBMs over shrunk bounds as mappings from $\mathcal{X}_0\times\mathcal{X}_0$ to shrunk bounds. They are called shrunk DBMs and, by Lemma 1.2, normalization is well-defined over them.

Usually, a shrunk DBM is encoded as a pair (M,P), where M is a DBM over $\mathcal{X}$, and P is a $|\mathcal{X}_0|\times|\mathcal{X}_0|$ non-negative integer matrix called a *shrinking matrix*. This pair represents the set of valuations defined by the DBM $M-\delta P$, for any given $\delta>0$. Considering the example $g=1+\delta<\mathsf{x}<2-\delta\wedge 2\delta<\mathsf{y}$, M is the guard obtained by setting $\delta=0$, and P is made of the integer multipliers of δ. Formally, given a fixed $\delta>0$ and a shrunk DBM (M,P) where $M(\mathsf{x},\mathsf{y})=(\prec_{\mathsf{x},\mathsf{y}},c_{\mathsf{x},\mathsf{y}})$ and $P(\mathsf{x},\mathsf{y})=p_{\mathsf{x},\mathsf{y}}$, $M-\delta P$ contains every valuation that satisfies

$$\bigwedge_{\mathsf{x},\mathsf{y}\in\mathcal{X}_0}\mathsf{x}-\mathsf{y}\prec_{\mathsf{x},\mathsf{y}}c_{\mathsf{x},\mathsf{y}}-\delta\,p_{\mathsf{x},\mathsf{y}}\,.$$

We adopt the following notation: when we write a statement involving a shrunk DBM (M,P), we mean that for some $\delta_0>0$, the statement holds for $M-\delta P$ for all $\delta\in(0,\delta_0]$. For instance, $(M,P)=\mathsf{PreTime}_{>\delta}((N,Q))$ means that $M-\delta P=\mathsf{PreTime}_{>\delta}(N-\delta Q)$ for all small enough $\delta>0$.

Shrunk DBMs are closed under standard operations on zones, and as a consequence, the CPre operator can be computed on shrunk DBMs:

Lemma 4.1 ([SBMR13])**.** *Let $e=(\ell,g,\mathcal{Y},\ell')$ be an edge and (M,P) be a shrunk DBM. Then, there exists a shrunk DBM (N,Q), that we can compute in polynomial time, such that $(N,Q)=\mathsf{CPre}_e^\delta((M,P))$.*

A shrunk DBM in normal form is empty if and only if it contains a bound stronger than $((\leqslant,0),0)$ in its diagonal entries. One could therefore decide if $r\cap\mathsf{CPre}_{\mathsf{p}}^\delta(r')\neq\emptyset$ for some $\delta>0$ by using shrunk DBMs. In this chapter, they are used as theoretical tools, because an easier characterization of region paths that can be robustly followed exists, that we now present.

4.1.3. Non-punctual region path

A region is said to be *non-punctual* if it contains two valuations separated by a positive time delay. Otherwise it is said punctual. Consider a region path p, and recall that it is a sequence of transitions of the form $(\ell,r_0)\xrightarrow{r_1,e}(\ell',r_2)$ meaning that from r_0, a time elapse leads to r_1, from which edge e can be taken and projects r_1 into r_2. If r_1 is

punctual, then clearly p cannot be robustly followed, as any perturbation, however small, will force controller out of p. When every region r_1 in p reached after a time elapse is non-punctual, we say that p is a non-punctual region path. In [SBMR13], it is shown that this condition is sufficient for finite paths:

Proposition 4.1 ([SBMR13]). *A finite region path can be robustly followed if and only if it is non-punctual.*

By using non-punctuality one can show that the qualitative robust controller synthesis problem over a reachability condition is PSPACE-complete. However, a Büchi condition requires robustness over an infinite path, and non-punctuality is no longer enough in this case.

4.2. Aperiodic cycles

Consider a finite region path p that forms a cycle, *i.e.* it ends in the same region state (ℓ, r) it started in. We want to know if p can be robustly iterated, meaning that there exists a zone of lower granularity $Z \subseteq r$ such that $Z \neq 0$ and $Z \subseteq \mathsf{CPre}^{\delta}_{\mathsf{p}}(Z)$. In particular, this implies that p^i can be robustly followed for every $i > 0$. The reachability relation of a region path p, denoted by $\mathsf{Reach}(\mathsf{p})$ is the set of pairs $(\nu, \nu') \in r \times r$ such that there exists an execution starting from (ℓ, ν) and ending in (ℓ, ν') that follows p. We say that the reachability relation $\mathsf{Reach}(\mathsf{p})$ is complete if $\mathsf{Reach}(\mathsf{p}) = r \times r$.

Definition 4.3. A region cycle p is *aperiodic* if for some $k \geqslant 1$, the reachability relation along the region path p^k is complete.

Proposition 4.2 ([SBMR13]). *A region cycle is robustly iterable if and only if it is non-punctual and* aperiodic

Moreover, aperiodicity can be checked for by using a dedicated structure called a *folded orbit graph*. The *orbit graph* of a region path p is a bipartite graph whose vertices are the corners of the first and last regions of the path; there is an edge between two corners if there exists an execution following p whose starting and ending valuations are arbitrarily close to the corners. When the region path is a cycle, the folded orbit graph is obtained by superposing initial and final vertices. This folded orbit graph is *aperiodic* if and only if for some $k \geqslant 1$, the sequence p^k has a folded orbit graph that is a complete graph. This should happen for some k at most exponential in $|\mathcal{X}|$ [SBMR13], and thus aperiodicity of a region path can be checked in polynomial space.

Overall, the qualitative robust controller synthesis problem with a Büchi condition can be solved by looking for a non-punctual aperiodic winning lasso $\mathsf{p}_0\mathsf{p}^\omega$ in the region abstraction, *i.e.* a non-punctual region paths p_0 from the initial region state $(\ell_0, \mathbf{0})$ to a target region state (ℓ_t, r) with $\ell_\mathsf{t} \in L_\mathsf{t}$ and a non-punctual aperiodic region cycle p around (ℓ_t, r). Using a variant of this procedure, where the winning lasso is guessed and aperiodicity is checked for with folded orbit graphs, it is shown in [SBMR13] that the problem is PSPACE-complete.

4.3. Generalization from region paths to paths

Definition 4.4. A finite path π is called a *progress cycle* if it is a cycle and if every clock is reset in at least one of its edges, *i.e.* $\mathcal{X} = \bigcup_{(\ell,g,\mathcal{Y},\ell')\in\pi} \mathcal{Y}$.

If a region cycle p following π is aperiodic, π must be a progress cycle, as otherwise any clock that is never reset in π will increase strictly in value while following p and the reachability relation will not be complete for any p^k. This holds because we did not authorize null delays.

In this section, our goal is to extend the previous results and characterize games where controller has a winning strategy in $\mathcal{G}_\delta(\mathcal{A})$ by a property expressed on the reachability relation of paths in $\mathcal{A}$, as opposed to the reachability relation of region paths. The reachability relation of a path π that starts in ℓ and ends in ℓ', denoted by $\mathsf{Reach}(\pi)$, is the set of pairs (ν, ν') such that there is an execution of $[\![\mathcal{A}]\!]$ starting from (ℓ, ν) and ending in (ℓ', ν') that follows π.

We will prove the following result:

Lemma 4.2. *There exists $\delta > 0$ such that controller has a winning strategy in $\mathcal{G}_\delta(\mathcal{A})$ if and only if there exist two paths π_1 and π_2 in $\mathcal{A}$, and a region r in $\mathbb{R}^{\mathcal{X}}_{\geqslant 0,<M}$, such that:*

- *π_1 is a path from ℓ_0 to some accepting location ℓ_t;*
- *π_2 is a progress cycle around ℓ_t;*
- *there exists a non-punctual region path from $(\ell_0, \mathbf{0})$ to (ℓ_t, r) along π_1;*
- *there exists a non-punctual region cycle around (ℓ_t, r) along π_2;*
- *the reachability relation of π_2 is complete over r, i.e. $r \times r \subseteq \mathsf{Reach}(\pi_2)$.*

The main difference with previous results is that π_2 can contain several region cycles around r, and instead of asking one of them to be aperiodic, we only ask that the entire reachability relation is complete, and thus that the union of the reachability relations of all such region cycles covers $r \times r$.

Proof of Lemma 4.2, direction ($\Rightarrow$). As previously explained, if $\mathcal{A}$ satisfies the robust controller synthesis problem, then there exists a non-punctual region path $\mathsf{p}_1\mathsf{p}_2$ in the region automaton of $\mathcal{A}$, such that p_1 reaches a target region (ℓ_t, r) from $(\ell_0, \mathbf{0})$, and p_2 is an aperiodic cycle around (ℓ_t, r). Let π_1 (resp. π_2) be a path that contains p_1 (resp. p_2). There exists k such that the reachability relation of p_2^k is complete, and therefore $r \times r \subseteq \mathsf{Reach}(\pi_2^k)$. □

In order to prove the converse, we will rely on intermediate results from [SBMR13].

Lemma 4.3 (Corollary 2, [SBMR13])**.** *Let π be a path in $\mathcal{A}$. Let M and N be non-empty zones such that $N = \mathsf{CPre}^0_\pi(M)$. Then, for every shrinking matrix P there exists a shrinking matrix Q such that $\mathsf{CPre}^\delta_\pi(M - \delta P) = N - \delta Q$ holds for all small enough $\delta > 0$.*

Let $\mathcal{B}_\infty(\nu, \varepsilon)$ describes $\{\nu' \mid \forall \mathsf{x} \in \mathcal{X}, |\nu(\mathsf{x}) - \nu'(\mathsf{x})| < \varepsilon\}$. The following lemma states that one can find a sub-zone (zone of smaller granularity) with non-empty interior inside every non-empty zone.

Lemma 4.4 (Lemma 14, [SBMR13]). *For all non-empty zones M, there exists a non-empty set of valuations M' (described as a zone of smaller granularity, with constraints over $\mathbb{Q}$), a valuation $\nu \in M'$ and $\varepsilon > 0$ such that $\mathcal{B}_\infty(\nu, \varepsilon) \cap M \subseteq M'$, and for all shrinking matrices P such that $M - \delta P$ is non-empty for some $\delta > 0$, $M' \subseteq M - \delta' P$ for all small enough $\delta' > 0$.*

And the set of controllable predecessors of such a sub-zone is non-empty for small enough $\delta > 0$.

Lemma 4.5 (Lemma 12, [SBMR13]). *Let p be a non-punctual path ending in region r. Let $r' \subseteq r$ be a set of valuations such that there exists $\nu \in r'$ and $\varepsilon > 0$ with $\mathcal{B}_\infty(\nu, \varepsilon) \cap r \subseteq r'$. Then, $\mathsf{CPre}^\delta_{\mathsf{p}}(r')$ is non-empty for small enough $\delta > 0$.*

We can now finish the proof of Lemma 4.2.

Proof of Lemma 4.2, direction ($\Leftarrow$). Consider now two paths π_1 and π_2, and a non-punctual region lasso $\mathsf{p}_1\mathsf{p}_2^\omega$ around a region r as described in Lemma 4.2. Let us prove that controller has a winning strategy in $\mathcal{G}_\delta(\mathcal{A})$ for a small enough δ.

Let Z be the zone defined by r, such that $Z \times Z \subseteq \mathsf{Reach}(\pi_2)$. Let us apply Lemma 4.4 and let $Z' \subseteq Z$ be the zone of smaller granularity that is obtained. Since the reachability relation is complete over Z, we have $Z \subseteq \mathsf{CPre}^0_{\pi_2}(Z')$. This implies that there exists Z'' and P such that $Z \subseteq Z''$ and $Z'' - \delta P = \mathsf{CPre}^\delta_{\pi_2}(Z')$ for δ small enough, by Lemma 4.3. Moreover, by applying Lemma 4.5 on p_2, Z and Z', we get that $\mathsf{CPre}^\delta_{\mathsf{p}_2}(Z')$ is non-empty for small enough $\delta > 0$. By monotonicity of CPre, $\mathsf{CPre}^\delta_{\mathsf{p}_2}(Z') \subseteq \mathsf{CPre}^\delta_{\pi_2}(Z')$ (they are the same paths, except that guards are enlarged in π_2 when compared to p_2); hence $Z'' - \delta P$ is non-empty for small enough δ. By definition of Z', for small enough $\delta > 0$, $Z' \subseteq Z'' - \delta P$, therefore $Z' \subseteq \mathsf{CPre}^\delta_{\pi_2}(Z')$. This means that, for small enough δ, the controller can enforce staying in Z' by following π_2 as many times as he wants in the game $\mathcal{G}_\delta(\mathcal{A})$. Similarly, by applying Lemma 4.5 on p_1, Z and Z', we get that the initial configuration can ensure reaching Z' if δ is small enough. Hence, there exists $\delta > 0$ such that the controller has a winning strategy in $\mathcal{G}_\delta(\mathcal{A})$. □

5. A symbolic approach

In this chapter, we will describe a symbolic (*i.e.* zone-based) procedure solving the qualitative robust controller synthesis problem.

5.1. Reachability relation of a path

Before treating the robustness issues, we start by designing a symbolic (*i.e.* zone-based) approach to describe and compare the reachability relations of paths in timed automata. This will be crucial subsequently to design a termination criterion in the state space exploration of our robustness-checking algorithm.

The reachability relation $\mathsf{Reach}(\pi)$ of a path π can be seen as a subset of $\mathbb{R}_{\geqslant 0}^{\mathcal{X} \uplus \mathcal{X}'}$ where $\mathcal{X}'$ are primed versions of the clocks, such that each $(\nu, \nu') \in \mathsf{Reach}(\pi)$ if and only if there is an execution from valuation ν to valuation ν' following π. Unfortunately, reachability relations $\mathsf{Reach}(\pi)$ are not zones in general, that is, they cannot be represented using only atomic constraints over $\mathcal{X} \uplus \mathcal{X}'$. In fact, we shall see shortly that constraints of the form $\mathsf{x} - \mathsf{y} + \mathsf{u} - \mathsf{v} \leqslant c$ also appear, as already observed in [QSW17]. We thus cannot rely directly on the traditional difference bound matrices (DBMs) used to represent zones. We instead rely on the constraint graphs that were introduced in [CLJ99], and explored in [JR11] for the parametric case (the latter work considers enlarged constraints, and not shrunk ones as we study here). Our contribution is to use these graphs to obtain a syntactic check of inclusion of the according reachability relations.

5.1.1. Constraint graphs

Rather than considering the values of the clocks in $\mathcal{X}$, this data structure considers the date X_i of the latest reset of the clock x_i, and uses a new variable τ denoting the global timestamp. Note that the clock values can be recovered easily since $\mathsf{X}_i = \tau - \mathsf{x}_i$. For the extra clock x_0, we introduce variable X_0 equal to the global timestamp τ (since x_0 must remain equal to 0). A constraint graph defining a zone is a weighted transition system whose states are $\mathsf{X} = \{\mathsf{X}_0, \mathsf{X}_1, \ldots, \mathsf{X}_n\}$ labelled over the tropical semiring of bounds over $\mathbb{Z}$. This transition system is always a complete graph, *i.e.* it contains exactly one transition from X to Y for each pair of states (X, Y). We refer to its states as nodes, to its labels as weights and to its transitions as edges. Weights in the graph are thus pairs of the form $(\prec, c)$ (see Chapter 2.2 for details). Constraints on clocks are represented by weights on edges in the graph: a constraint $\mathsf{X} - \mathsf{Y} \prec c$ is represented by an edge from X to Y weighted by $(\prec, c)$, with $\prec \in \{<, \leqslant\}$ and $c \in \mathbb{Z}$. Weights $(<, +\infty)$ and $(<, -\infty)$ are also allowed for trivial constraints. Therefore, we can compute shortest weights between two

vertices of a weighted graph. A cycle is said to be negative if it has weight at most $(<, 0)$, *i.e.* $(<, 0)$ or $(\prec, c)$ with $c < 0$.

5.1.2. Encoding paths

Constraint graphs can also encode tuples of valuations seen along a path. To encode a k-step computation, we make $k+1$ copies of the nodes, that is, $(\mathsf{X}^i) = \{\mathsf{X}_0^i, \mathsf{X}_1^i, \ldots, \mathsf{X}_n^i\}$ for $i \in \{1, \ldots, k+1\}$. These copies are also called *layers*. Let us first consider an example on the path π consisting of the edge from ℓ_1 to ℓ_2, and the edge from ℓ_2 to ℓ_1, in the timed automaton of Figure 3.1. The constraint graph G_π is depicted in Figure 5.1: in our diagrams of constraint graphs, the absence of labels on an edge means $(\leqslant, 0)$, we depict with an edge with arrows on both ends the presence of an edge in both directions. We always represent with dashed arrows edges that are labelled by $(<, c)$, and plain arrows edges that are labelled with $(\leqslant, c)$, and the absence of an edge means that it is labelled with $(<, +\infty)$. The graph has five columns, each containing copies of the variables for that step: they represent the valuations before the first edge, after the first time elapse, after the first reset, after the second time elapse and after the second reset. In general now, each elementary operation can be described by a constraint graph with two layers $(\mathsf{X}_i)_{0 \leqslant i \leqslant n}$ (before) and $(\mathsf{X}'_i)_{0 \leqslant i \leqslant n}$ (after).

- The operation $\mathsf{Pretime}_{>t}$ is described by the constraint graph $G_{\mathsf{time}}^{>t}$ with edges $\mathsf{X}_i \to \mathsf{X}_0$, $\mathsf{X}_i \leftrightarrow \mathsf{X}'_i$ for $i > 0$, and $\mathsf{X}_0 \xrightarrow{(<,-t)} \mathsf{X}'_0$. Figure 5.1 contains two occurrences of $G_{\mathsf{time}}^{>0}$.
- The operation $[\![g]\!] \cap \mathsf{Unreset}_{\mathcal{Y}}(\cdot)$, to test a guard g and reset the clocks in $\mathcal{Y}$, is described by the constraint graph $G_{\mathsf{edge}}^{g,\mathcal{Y}}$ with edges $\mathsf{X}_0 \leftrightarrow \mathsf{X}'_0$ (meaning that the time does not elapse), $\mathsf{X}_i \leftrightarrow \mathsf{X}'_i$ for i such that clock $\mathsf{x}_i \notin \mathcal{Y}$, and $\mathsf{X}'_i \leftrightarrow \mathsf{X}'_0$ for i such that clock $\mathsf{x}_i \in \mathcal{Y}$, and for every constraint[1] $\mathsf{x}_i - \mathsf{x}_j \prec c$ appearing in g, an edge from X_j to X_i labelled by $(\prec, c)$ (since it encodes the fact that $(\tau - \mathsf{X}_i) - (\tau - \mathsf{X}_j) = \mathsf{X}_j - \mathsf{X}_i \prec c$). In Figure 5.1, we have first $G_{\mathsf{edge}}^{\mathsf{x}_1 \leqslant 2, \{\mathsf{x}_1\}}$, and then $G_{\mathsf{edge}}^{\mathsf{x}_2 \geqslant 2, \{\mathsf{x}_2\}}$.

Constraint graphs can be stacked one after the other to obtain the constraint graph of an edge e, and then of a path π, that we denote by G_π. In the resulting graph, there is one leftmost layer of vertices $(\mathsf{X}_i^{\boldsymbol{\ell}})_i$ and one rightmost one $(\mathsf{X}_i^{\mathbf{r}})_i$ representing the situation before and after the firing of the path π. Once this graph is constructed, the intermediary levels can be eliminated after replacing each edge between the nodes of $\mathsf{X}^{\boldsymbol{\ell}} \uplus \mathsf{X}^{\mathbf{r}}$ by the shortest path in the graph. This phase is hereafter called *normalisation* of the constraint graph. The normalised version of the constraint graph of Figure 5.1 is depicted on its right.

Constraint graphs can be encoded as DBMs over the set of clocks $\{\mathsf{X}_j^i \mid 0 \leqslant i \leqslant k, 0 \leqslant j \leqslant n\}$, and their normalization can therefore be obtained efficiently using DBM normalization. The normalization of G_π can be computed in time $\mathcal{O}(|\mathcal{X}|^3|\pi|)$ by following

[1] guards are seen as conjunctions of difference bound constraints by Lemma 2.1, with exactly one constraint for every pair of clocks in $\mathcal{X}_0$

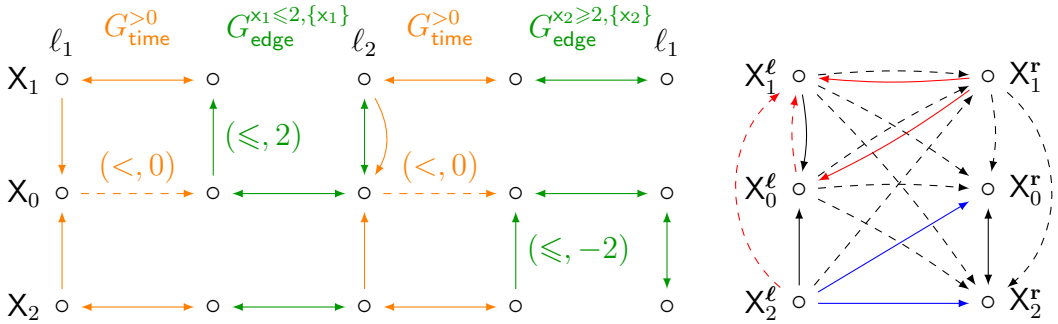


Figure 5.1.: On the left, the constraint graph of the path $\ell_1 \xrightarrow{\mathsf{x}_1\leqslant 2,\mathsf{x}_1:=0} \ell_2 \xrightarrow{\mathsf{x}_2\geqslant 2,\mathsf{x}_2:=0} \ell_1$. Orange and green colors are used to differentiate successive steps. On the right, its normalised version: dashed edges have weight $(<,.)$, plain edges have weight $(\leqslant,.)$, black edges have weight $(.,0)$, red edges have weight $(.,2)$ and blue edges have weight $(.,-2)$. For every node X, the normalized version also contains a loop $\mathsf{X} \xrightarrow{(\leqslant,0)} \mathsf{X}$ that is not displayed.

π one edge at a time and normalizing at each step. If the constraint graph G_π contains a negative cycle, then the zone associated with the DBM encoding it is empty, but that information may be lost when we remove the intermediary levels. In this case, we set the weight of the edges $(\mathsf{X}^{\boldsymbol{\ell}}_0, \mathsf{X}^{\boldsymbol{\ell}}_0)$ and $(\mathsf{X}^{\mathbf{r}}_0, \mathsf{X}^{\mathbf{r}}_0)$ in the normalized constraint graph to $(<,0)$.

5.1.3. From constraint graphs to reachability relations

From a logical point of view, the elimination of intermediary layers reflects an elimination of quantifiers in a formula of the first-order theory of real numbers. At the end, we obtain a set of constraints of the form $\mathsf{X}^{\mathbf{k}}_i - \mathsf{X}^{\mathbf{k}'}_j \prec c$ with $\mathbf{k}, \mathbf{k}' \in \{\boldsymbol{\ell}, \mathbf{r}\}$. These constraints do not reflect uniquely the reachability relation $\mathsf{Reach}(\pi)$, in the sense that it is possible that $\mathsf{Reach}(\pi_1) = \mathsf{Reach}(\pi_2)$ but the normalised versions of G_{π_1} and G_{π_2} are different. For example, if we consider the path π^2 obtained by repeating twice the cycle π between ℓ_1 and ℓ_2, the reachability relation does not change ($\mathsf{Reach}(\pi^2) = \mathsf{Reach}(\pi)$), but the normalised constraint graph does ($G_{\pi^2} \neq G_\pi$): all labels $(\leqslant,2)$ of the red dotted edges from the rightmost layer to the leftmost layer become $(\leqslant,4)$, and the labels $(\leqslant,-2)$ of the dashed blue edges become $(\leqslant,-4)$.

We solve this issue by jumping back from variables $\mathsf{X}^{\mathbf{k}}_i$ to the clock valuations. Indeed, in terms of clock valuations $\nu^{\boldsymbol{\ell}}$ and $\nu^{\mathbf{r}}$ before and after the path, the constraint $\mathsf{X}^{\mathbf{k}}_i - \mathsf{X}^{\mathbf{k}'}_j \prec c$ (for $\mathbf{k}, \mathbf{k}' \in \{\boldsymbol{\ell}, \mathbf{r}\}$) rewrites as

$$(\tau^{\mathbf{k}} - \nu^{\mathbf{k}}(\mathsf{x}_i)) - (\tau^{\mathbf{k}'} - \nu^{\mathbf{k}'}(\mathsf{x}_j)) \prec c\,,$$

where $\tau^{\boldsymbol{\ell}}$ is the global timestamp before firing π and $\tau^{\mathbf{r}}$ the one after. When $\mathbf{k} = \mathbf{k}'$, variables $\tau^{\boldsymbol{\ell}}$ and $\tau^{\mathbf{r}}$ disappear, leaving a constraint of the form

$$\nu^{\mathbf{k}}(\mathsf{x}_j) - \nu^{\mathbf{k}}(\mathsf{x}_i) \prec c\,.$$

When $\mathbf{k} \neq \mathbf{k}'$, we can rewrite the constraint as

$$\tau^{\mathbf{k}} - \tau^{\mathbf{k}'} \prec \nu^{\mathbf{k}}(\mathsf{x}_i) - \nu^{\mathbf{k}'}(\mathsf{x}_j) + c\,.$$

We therefore obtain upper and lower bounds on the value of $\tau^{\mathbf{r}} - \tau^{\boldsymbol{\ell}}$, allowing us to eliminate $\tau^{\mathbf{r}} - \tau^{\boldsymbol{\ell}}$, considered as a single variable that contains the time elapsed during the firing of π. We therefore obtain *in fine* a formula mixing constraints of the form

- $\nu^{\mathbf{k}}(\mathsf{x}_a) - \nu^{\mathbf{k}}(\mathsf{x}_b) \prec p$, with $\mathbf{k} \in \{\boldsymbol{\ell}, \mathbf{r}\}$, $a \neq b$, and we define $\gamma^{\mathbf{k}}_{a,b} = (\prec, p)$;
- $\nu^{\boldsymbol{\ell}}(\mathsf{x}_a) - \nu^{\boldsymbol{\ell}}(\mathsf{x}_b) + \nu^{\mathbf{r}}(\mathsf{x}_c) - \nu^{\mathbf{r}}(\mathsf{x}_d) \prec p$, with $a \neq b$ and $c \neq d$, and we define $\gamma_{a,b,c,d} = (\prec, p)$. This constraint can appear in two ways: either from

  $$\nu^{\mathbf{r}}(\mathsf{x}_c) - \nu^{\boldsymbol{\ell}}(\mathsf{x}_b) + p_1 \prec_1 \tau^{\mathbf{r}} - \tau^{\boldsymbol{\ell}} \prec_2 \nu^{\mathbf{r}}(\mathsf{x}_d) - \nu^{\boldsymbol{\ell}}(\mathsf{x}_a) + p_2$$

  by eliminating $\tau^{\mathbf{r}} - \tau^{\boldsymbol{\ell}}$, or by adding the two constraints of the form

  $$\nu^{\boldsymbol{\ell}}(\mathsf{x}_a) - \nu^{\boldsymbol{\ell}}(\mathsf{x}_b) \prec_1 p_1 \text{ and } \nu^{\mathbf{r}}(\mathsf{x}_c) - \nu^{\mathbf{r}}(\mathsf{x}_d) \prec_2 p_2\,.$$

  Thus, $\gamma_{a,b,c,d}$ is obtained as the strongest of the two constraints obtained in this manner. In other terms, in the constraint graph, this constraint is the minimal weight between the sum of the weights of the edges $(\mathsf{X}^{\mathbf{r}}_d, \mathsf{X}^{\boldsymbol{\ell}}_a)$ and $(\mathsf{X}^{\boldsymbol{\ell}}_b, \mathsf{X}^{\mathbf{r}}_c)$, and the sum of the weights of the edges $(\mathsf{X}^{\boldsymbol{\ell}}_b, \mathsf{X}^{\boldsymbol{\ell}}_a)$ and $(\mathsf{X}^{\mathbf{r}}_d, \mathsf{X}^{\mathbf{r}}_c)$. For example, in the path in Figure 5.1, we have $\gamma_{0,1,0,2} = (\leqslant, 0)$ since the two weights are respectively $(<, +\infty)$ and $(\leqslant, 0)$, whereas $\gamma_{1,2,2,1} = (\leqslant, 0)$ because the two weights are respectively $(\leqslant, 0)$ and $(<, 2)$.

Let $\varphi(G)$ be the conjunction of such constraints, obtained from the normalization of a constraint graph G: this is a quantifier-free formula of the additive theory of reals. We obtain the following property, whose proof can be derived from the developments of [CLJ99].

Lemma 5.1. *Let π be a path in a timed automaton. $\mathsf{Reach}(\pi)$ is the set of pairs of valuations $(\nu^{\boldsymbol{\ell}}, \nu^{\mathbf{r}})$ that satisfy the formula $\varphi(G_\pi)$.*

Proof. As we have seen, the constraints of G_π encode syntactically the constraints that valuations must meet to form an execution following π. Normalization of DBMs leaves the represented zone unchanged, and therefore $\varphi(G_\pi)$ is a formula that describes $\mathsf{Reach}(\pi)$. A particular case is the one where $\mathsf{Reach}(\pi) = \emptyset$, where we made sure that $\varphi(G_\pi)$ contains the constraint $\nu^{\boldsymbol{\ell}}(\mathsf{x}_0) - \nu^{\boldsymbol{\ell}}(\mathsf{x}_0) < 0$ and is therefore not satisfiable. □

5.1.4. Checking inclusion

Checking formulæ like $\varphi(G_\pi)$ (*i.e.* conjunctions of linear inequalities) for inclusion is polynomial in general, as one can use linear programming techniques. In this case however, a more efficient way exists.

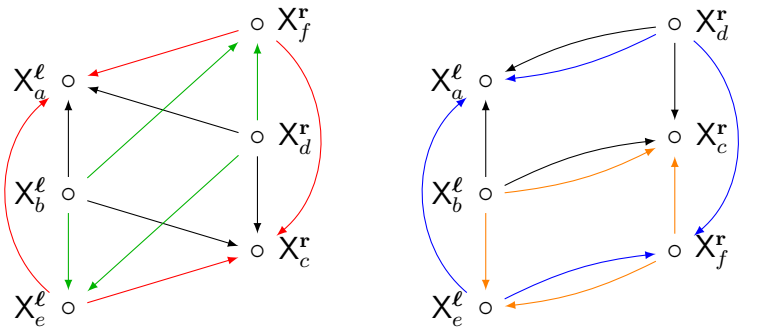


Figure 5.2.: Normalised constraint graphs involved in a possible deduction of stronger constraints.

For a path π, we regroup the pairs $(\gamma^{\ell}_{a,b})$, $(\gamma^{\mathbf{r}}_{a,b})$ and $(\gamma_{a,b,c,d})$ above in a single vector Γ^{π} (of size $\mathcal{O}(|\mathcal{X}|^4)$). We extend the comparison relation $\leqslant$ to these vectors by applying it componentwise. These vectors can be used to check equality or inclusion of reachability relations in time $\mathcal{O}(|\mathcal{X}|^4)$:

Theorem 5.1. *Let π and π' be paths in a timed automaton such that* $\mathsf{Reach}(\pi)$ *and* $\mathsf{Reach}(\pi')$ *are non empty. Then,* $\mathsf{Reach}(\pi) \subseteq \mathsf{Reach}(\pi')$ *if and only if* $\Gamma^{\pi} \leqslant \Gamma^{\pi'}$.

Proof. The main argument on why the syntactic check on $\varphi(G_\pi)$ is sufficient amounts directly from the normalisation of the constraint graph. Indeed, let us show that the constraints obtained in a formula $\varphi(G_\pi)$ are the strongest possible. For that, suppose that we obtain a stronger constraint on $\nu^{\ell}(\mathsf{x}_a) - \nu^{\ell}(\mathsf{x}_b) + \nu^{\mathbf{r}}(\mathsf{x}_c) - \nu^{\mathbf{r}}(\mathsf{x}_d)$ by combining two inequalities of $\varphi(G_\pi)$ (it is then possible to extend the argument to work for all positive linear combinations of inequalities): they must then be either of the form

$$\begin{cases} \nu^{\ell}(\mathsf{x}_a) - \nu^{\ell}(\mathsf{x}_e) + \nu^{\mathbf{r}}(\mathsf{x}_c) - \nu^{\mathbf{r}}(\mathsf{x}_f) \prec_1 p_1 \\ \nu^{\ell}(\mathsf{x}_e) - \nu^{\ell}(\mathsf{x}_b) + \nu^{\mathbf{r}}(\mathsf{x}_f) - \nu^{\mathbf{r}}(\mathsf{x}_d) \prec_2 p_2 \end{cases} \quad \text{or} \quad \begin{cases} \nu^{\ell}(\mathsf{x}_a) - \nu^{\ell}(\mathsf{x}_e) + \nu^{\mathbf{r}}(\mathsf{x}_f) - \nu^{\mathbf{r}}(\mathsf{x}_d) \prec_1 p_1 \\ \nu^{\ell}(\mathsf{x}_e) - \nu^{\ell}(\mathsf{x}_b) + \nu^{\mathbf{r}}(\mathsf{x}_c) - \nu^{\mathbf{r}}(\mathsf{x}_f) \prec_2 p_2 \end{cases}$$

Summing up each pair gives a new possible constraint on $\nu^{\ell}(\mathsf{x}_a) - \nu^{\ell}(\mathsf{x}_b) + \nu^{\mathbf{r}}(\mathsf{x}_c) - \nu^{\mathbf{r}}(\mathsf{x}_d)$. This corresponds to the two situations in the constraint graphs depicted in Figure 5.2. Using the normalisation of the constraint graphs implies then easily that the new constraints are weaker than the original constraint already present in Γ^{π}. □

Notice that we do not need to check equivalence or implication of formulæ $\varphi(G_\pi)$ and $\varphi(G_{\pi'})$, but simply check syntactically constants appearing in these formulæ. Moreover, these constants can be extracted easily from the normalized constraint graph stored as a DBMs on $2 \times |\mathcal{X}_0|$ clocks, allowing for reusability of classical DBM libraries. For the constraint graph in Figure 5.1, we have seen that $G_{\pi^2} \neq G_\pi$, even if $\mathsf{Reach}(\pi^2) = \mathsf{Reach}(\pi)$. However, we can check that $\varphi(G_{\pi^2}) = \varphi(G_\pi)$ as expected.

5.1.5. Computation of Pre and Post

By Lemma 5.1 and the construction of constraint graphs, one can easily compute

$$\mathsf{Pre}_\pi(Z) = \{\nu \mid \exists \nu' \in Z, ((\ell,\nu),(\ell',\nu')) \in \mathsf{Reach}(\pi)\}$$

for a given path π and zone Z (see [CLJ99, JR11]). In fact, consider the normalised constraint graph G_π on nodes $\mathsf{X}^{\boldsymbol{\ell}} \cup \mathsf{X}^{\mathbf{r}}$. To compute $\mathsf{Pre}_\pi(Z)$, one just needs to add the constraints of Z on $\mathsf{X}^{\mathbf{r}}$. This is done by replacing each edge $\mathsf{X}^{\mathbf{r}}_i \xrightarrow{w} \mathsf{X}^{\mathbf{r}}_j$ by $\mathsf{X}^{\mathbf{r}}_i \xrightarrow{\min(Z_{j,i},w)} \mathsf{X}^{\mathbf{r}}_j$ where $Z_{j,i} = (\prec, p)$ defines the constraint of Z on $\mathsf{x}_j - \mathsf{x}_i$. Then, the normalisation of the graph describes the reachability relation along path π ending in zone Z. Furthermore, projecting the constraints to $\mathsf{X}^{\boldsymbol{\ell}}$ yields $\mathsf{Pre}_\pi(Z)$: this can be obtained by gathering all constraints on pairs of nodes of $\mathsf{X}^{\boldsymbol{\ell}}$. One can symmetrically compute the successor

$$\mathsf{Post}_\pi(Z) = \{\nu' \mid \exists \nu \in Z, ((\ell,\nu),(\ell',\nu')) \in \mathsf{Reach}(\pi)\}$$

by constraining the nodes $\mathsf{X}^{\boldsymbol{\ell}}$ and projecting to $\mathsf{X}^{\mathbf{r}}$.

5.2. Robust iterability of a lasso

In this section, we study the perturbation game $\mathcal{G}_\delta(\mathcal{A})$ between the two players (controller and environment), as defined in Chapter 3, when the timed automaton $\mathcal{A}$ is restricted to a fixed *lasso* $\pi_1\pi_2^\omega$, *i.e.* π_1 is a path from ℓ_0 to some accepting location ℓ_t, and π_2 a cyclic path around ℓ_t. This implies that the controller does not have the choice of the transitions, but only of the delays, while the environment only perturbates these delays. We will consider different settings, in which δ is fixed or not.

5.2.1. Controllable predecessors and their greatest fixpoints

We start by studying the controllable predecessor $\mathsf{CPre}^\delta_\pi(Z)$ defined in Section 4.1 as the set of valuations from which the controller can ensure reaching the set of valuations Z by following π.

This operator is monotone, hence its greatest fixpoint $\nu X\, \mathsf{CPre}^\delta_\pi(X)$ is well-defined, equal to $\bigcap_{i\geqslant 0} \mathsf{CPre}^\delta_{\pi^i}(\top)$: it corresponds to the valuations from which the controller can guarantee to loop forever along the path π. By definition of the game $\mathcal{G}_\delta(\mathcal{A})$ where $\mathcal{A}$ is restricted to the lasso $\pi_1\pi_2^\omega$, the controller wins the game if and only if $\mathbf{0} \in \mathsf{CPre}^\delta_{\pi_1}(\nu X\, \mathsf{CPre}^\delta_{\pi_2}(X))$. As a consequence, our problem reduces to the computation of this greatest fixpoint.

5.2.2. Branching constraint graphs

We consider first a fixed (rational) value of the parameter δ, and are interested in the computation of the greatest fixpoint $\nu X\, \mathsf{CPre}^\delta_{\pi_2}(X)$. In [JR11], constraints graphs were used to provide a termination criterion allowing one to compute the greatest fixpoint of

the classical predecessor operator CPre^0_π. We generalise this approach to deal with the operator CPre^δ_π and to this end, we need to generalise constraint graphs. Unfortunately, the operator $\mathsf{shrink}_{[-\delta,\delta]}$ cannot be encoded in a constraint graph. Intuitively, this comes from the fact that a constraint graph represents a relation between valuations, while there is no such relation associated with the CPre^δ_π operator. Instead, we introduce *branching constraint graphs* that will faithfully represent the CPre^δ_π operator: unlike constraint graphs introduced so far that have a left layer and a right layer of variables, a branching constraint graph has still a single left layer but several right layers. We also need weights in this new constraint graph to be rational numbers, such that if $N \in \mathbb{N}_{>0}$ is the smallest granularity with $\delta N \in \mathbb{N}$, then the branching constraint graph will be labelled over the tropical semiring of bounds over $\mathbb{Q}_N$.

We first define a branching constraint graph $G^\delta_{\mathsf{shrink}}$ associated with the operator $\mathsf{shrink}_{[-\delta,\delta]}$ as follows. Its set of vertices is composed of three copies of $\{\mathsf{X}_0, \mathsf{X}_1, \ldots, \mathsf{X}_n\}$, denoted by primed, unprimed and doubly primed versions. Edges are defined so as to encode the following constraints : $\mathsf{X}'_i = \mathsf{X}_i$ and $\mathsf{X}''_i = \mathsf{X}_i$ for every $i \neq 0$, and $\mathsf{X}'_0 = \mathsf{X}_0 + \delta$ and $\mathsf{X}''_0 = \mathsf{X}_0 - \delta$. An instance of this graph can be found in several occurrences in Figure 5.3.

Proposition 5.1. *Let Z be a zone and $G^\delta_{\mathsf{shrink}}(Z)$ be the graph obtained from $G^\delta_{\mathsf{shrink}}$ by adding on primed and doubly primed vertices the constraints defining Z (as for $\mathsf{Pre}_\pi(Z)$ in Section 5.1.5). Then the constraint on unprimed vertices obtained from the shortest paths in $G^\delta_{\mathsf{shrink}}(Z)$ is equivalent to $\mathsf{shrink}_{[-\delta,\delta]}(Z)$.*

Proof. Given a zone Z and a real number d, we define $Z + d = \{\nu + d \mid \nu \in Z\}$. One easily observes that $\mathsf{shrink}_{[-\delta,\delta]}(Z) = (Z + \delta) \cap (Z - \delta)$. The result follows from the observation that taking two distinct copies of vertices, and considering shortest paths allows one to encode the intersection. □

Then, for all edges $e = (\ell, g, \mathcal{Y}, \ell')$, we define the branching constraint graph G^δ_e as the graph obtained by stacking (in this order) the branching constraint graph $G^{>\delta}_{\mathsf{time}}$, $G^\delta_{\mathsf{shrink}}$ and $G^{g,\mathcal{Y}}_{\mathsf{edge}}$. Note that two copies of the graph $G^{g,\mathcal{Y}}_{\mathsf{edge}}$ are needed, to be connected to the two sets of vertices that are on the right of the graph $G^\delta_{\mathsf{shrink}}$. This definition is extended in the expected way to a finite path π, yielding the graph G^δ_π. In this graph, there is a single set of vertices on the left, and $2^{|\pi|}$ sets of vertices on the right. As a direct consequence of the previous results on the constraint graphs for time elapse, shrinking and guard/reset, one obtains:

Proposition 5.2. *Let Z be a zone and π be a path. We let $G^\delta_\pi(Z)$ be the graph obtained from G^δ_π by adding on every set of right vertices the constraints defining Z. Then the constraint on the left layer of vertices obtained from the shortest paths in $G^\delta_\pi(Z)$ is equivalent to $\mathsf{CPre}^\delta_\pi(Z)$.*

An example of the graph $G^\delta_\pi(Z)$ for $\pi = e_1 e_2$, edges considered in Figure 5.1, is depicted in Figure 5.3 (on the left).

We are now ready to prove the following result, generalisation of [JR11, Lemma 2], that will allow us to compute the greatest fixpoint of the operator CPre^δ_π:

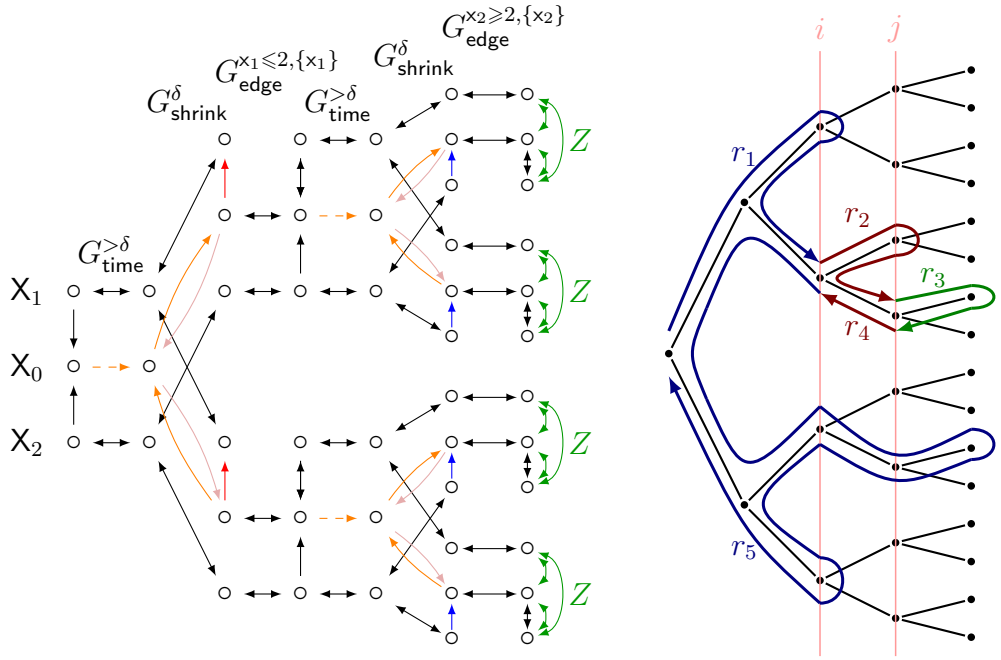


Figure 5.3.: On the left, the branching constraint graph $G^{\delta}_{e_1e_2}$ encoding the operator $\mathsf{CPre}^{\delta}_{e_1e_2}$, where e_1 and e_2 refer to edges considered in Figure 5.1. Dashed edges have weight $(<,.)$, plain edges have weight $(\leqslant,.)$. Black edges (resp. orange edges, pink edges, red edges, blue edges) are labelled by $(.,0)$ (resp. $(.,-\delta)$, $(.,\delta)$, $(.,2)$,$(.,-2)$). On the right, a decomposition of a path in a branching constraint graph G^{δ}_{π} (proof of Proposition 5.3).

Proposition 5.3. *Let π be a path and δ be a non-negative rational number. We let $N = |\mathcal{X}_0|^2$. If $\mathsf{CPre}^{\delta}_{\pi^{2N+1}}(\top) \subsetneq \mathsf{CPre}^{\delta}_{\pi^{2N}}(\top)$, then $\nu X\, \mathsf{CPre}^{\delta}_{\pi}(X) = \emptyset$.*

Proof. Assume $\mathsf{CPre}^{\delta}_{\pi^{2N+1}}(\top) \subsetneq \mathsf{CPre}^{\delta}_{\pi^{2N}}(\top)$ and consider the zones $\mathsf{CPre}^{\delta}_{\pi^{N+1}}(\top)$ (represented by the DBM M_1) and $\mathsf{CPre}^{\delta}_{\pi^{N}}(\top)$ (represented by the DBM M_2). We have $M_1 \subsetneq M_2$, as otherwise the fixpoint would have already been reached after N steps. By Proposition 5.2, the zone corresponding to M_1 is associated with shortest paths between vertices on the left in the graph $G^{\delta}_{\pi^{N+1}}$. In the sequel, given a path r in this graph, $w(r)$ denotes its weight. We distinguish two cases:

Case 1: $M_1 \subsetneq M_2$ because of the rational coefficients. Then, there exists an entry $(\mathsf{x},\mathsf{y}) \in \mathcal{X}_0^2$ such that $\mathsf{M}_1[\mathsf{x},\mathsf{y}] < \mathsf{M}_2[\mathsf{x},\mathsf{y}]$.[2] The value $M_1[\mathsf{x},\mathsf{y}]$ is thus associated with a shortest path between vertices X and Y in $G^{\delta}_{\pi^{N+1}}$. We fix a shortest path of minimal length, and denote it by r. As the entry is strictly smaller than in M_2, this shortest path should reach the last copy of the graph G^{δ}_{π}. This path can be interpreted as a traversal of the binary tree of depth $|\mathcal{X}_0|^2+1$, reaching at least one leaf. We can prove that this entails that there exists a pair of clocks $(\mathsf{u},\mathsf{v}) \in \mathcal{X}_0^2$ appearing at two levels $i < j$ of this tree, and

[2]Here we borrow notations from [JR11], where $\mathsf{M}_1[\mathsf{x},\mathsf{y}]$ refers to the constant c such that $M_1[\mathsf{x},\mathsf{y}] = (\prec, c)$.

a decomposition $r = r_1 r_2 r_3 r_4 r_5$ of the path, such that $w(r_2) + w(r_4) = (\prec, d)$ with $d < 0$ (Property (†)). In addition, in this decomposition, r_3 is included in subgraphs of levels $k \geq j$, and the pair of paths (r_2, r_4) is called a *return path*, following the terminology of [JR11]. This decomposition is depicted in Figure 5.3 (on the right). Intuitively, the property (†) follows from the fact that as r_3 is included in subgraphs of levels $k \geq j$, and because the final zone (on the right) is the zone $\top$ which adds no edges, the concatenation $r' = r_1 r_3 r_5$ is also a valid path from X to Y in $G^{\delta}_{\pi^{N+1}}$, and is shorter than r. We conclude using the fact that r has been chosen as a shortest path of minimal weight.

Property (†) allows us to prove that the greatest fixpoint is empty. Indeed, by considering iterations of π, one can repeat the return path associated with (r_2, r_4) and obtain paths from X to Y whose weights diverge towards $-\infty$.

Case 2: $M_1 \subsetneq M_2$ because of the ordering coefficients. We claim that this case cannot occur. Indeed, one can show that the constants will not evolve anymore after the Nth iteration of the fixpoint: the coefficients can only decrease by changing from a non-strict inequality $(\leq, c)$ to a strict one $(<, c)$. This propagation of strict inequalities is performed in at most $|\mathcal{X}_0|^2$ additional steps, thus we have $\mathsf{CPre}^{\delta}_{\pi^{2N+1}}(\top) = \mathsf{CPre}^{\delta}_{\pi^{2N}}(\top)$, yielding a contradiction. □

Compared to the result of [JR11], the number of iterations needed before convergence grows from $|\mathcal{X}_0|^2$ to $2|\mathcal{X}_0|^2$: this is due to the presence of strict and non-strict inequalities, not considered in [JR11]. With the help of branching constraint graphs, we have thus shown that the greatest fixpoint can be computed in finite time: this can then be done directly with computations on zones (and not on branching constraint graphs).

Proposition 5.4. *Given a path π and a rational number δ, the greatest fixpoint $\nu X\, \mathsf{CPre}^{\delta}_{\pi}(X)$ can be computed in time polynomial in $|\mathcal{X}|$ and $|\pi|$. As a consequence, one can decide whether the controller has a winning strategy along a lasso $\pi_1 \pi_2^{\omega}$ in $\mathcal{G}_{\delta}(\mathcal{A})$ in polynomial time.*

5.2.3. Solving the qualitative problem for a lasso

We have shown how to decide whether the controller has a winning strategy for a fixed rational value of δ. We now aim at deciding whether there exists a positive value of δ for which the controller wins the game $\mathcal{G}_{\delta}(\mathcal{A})$ (where $\mathcal{A}$ is restricted to a lasso $\pi_1 \pi_2^{\omega}$). To this end, we will use the shrunk DBMs of Section 4.1.

Proposition 5.5. *Given a path π, one can compute a shrunk DBM (M, P) equal to the greatest fixpoint of the operator $\mathsf{CPre}^{\delta}_{\pi}$. As a consequence, one can solve the qualitative robust controller synthesis problem for a given lasso in time complexity polynomial in the number of clocks and in the length of the lasso.*

Proof. The bound $2|\mathcal{X}_0|^2$ identified in Proposition 5.3 for the greatest fixpoint does not depend on the value of δ. Hence the algorithm for computing a shrunk DBM representing the greatest fixpoint proceeds as follows. It computes symbolically, using shrunk DBMs, the $2|\mathcal{X}_0|^2$-th and $2|\mathcal{X}_0|^2 + 1$-th iterations of the operator $\mathsf{CPre}^{\delta}_{\pi}$, from the zone $\top$. By

monotonicity, the $2|\mathcal{X}_0|^2+1$-th iteration is included in the $2|\mathcal{X}_0|^2$-th. If the two shrunk DBMs are equal, then they are also equal to the greatest fixpoint. Otherwise, the greatest fixpoint is empty. To decide the qualitative robust controller synthesis problem for a given lasso, one first computes a shrunk DBM representing the greatest fixpoint associated with π_2 and, if not empty, one computes a new shrunk DBM by applying to it the operator $\mathsf{CPre}^{\delta}_{\pi_1}$. Then, one checks whether the valuation $\mathbf{0}$ belongs to the zone represented by the resulting shrunk DBM. □

5.3. Synthesis of robust controllers

We are now ready to solve the qualitative robust controller synthesis problem on all timed automata, that is to find, if it exists, a lasso $\pi_1\pi_2^\omega$ and a perturbation δ such that the controller wins the game $\mathcal{G}_\delta(\mathcal{A})$ when following the lasso $\pi_1\pi_2^\omega$ as a strategy. As for the symbolic checking of emptiness of a Büchi timed language [LOD+13], we will use a nested forward analysis to exhaust all possible lassos, each being tested for robustness by the techniques studied in previous section: a first forward analysis will search for π_1, a path from the initial location to an accepting location, and a second forward analysis from each accepting location ℓ_t to find a cycle π_2, not necessarily simple, around ℓ_t. Forward analysis means that we compute the successor zone $\mathsf{Post}_\pi(Z)$ when following path π from zone Z: it is described formally in Chapter 2.3.5 as a symbolic algorithm solving the emptiness problem for reachability conditions.

5.3.1. Abstraction of lassos

Before studying in more details the two independant forward analyses, we first study what information we must keep about π_1 and π_2 in order to still being able to test the robustness of the lasso $\pi_1\pi_2^\omega$. A classical problem for robustness is the firing of a *punctual edges*, *i.e.* an edge where controller has a single choice of time delay: clearly such a firing will be robust for no possible choice of parameter δ. Therefore, we must at least forbid such punctual edges in our forward analyses. We thus introduce a non-punctual successor operator $\mathsf{Post}^{\mathrm{np}}_\pi$. It consists of the standard successor operator Post_π in the timed automaton $\mathcal{A}^{\mathrm{np}}$ obtained from $\mathcal{A}$ by making strict every inequality symbol in guards ($1 \leqslant \mathsf{x} \leqslant 2$ becomes $1 < \mathsf{x} < 2$). The crucial point is that if a positive delay d can be taken by the controller while satisfying a set of strict constraints, then other delays are also possible, close enough to d. By analogy, recall that a region is said to be *non-punctual* if it contains two valuations separated by a positive time delay. In particular, if such a region satisfies a non-diagonal atomic constraint in $\mathcal{A}$ it also satisfies the corresponding strict constraint in $\mathcal{A}^{\mathrm{np}}$.

Controller wins $\mathcal{G}_\delta(\mathcal{A})$ for some $\delta > 0$ if and only if he wins $\mathcal{G}_\delta(\mathcal{A}^{\mathrm{np}})$ for some $\delta > 0$. This observation can be derived from [SBMR13], as every non-punctual lasso in the region abstraction of $\mathcal{A}$ is a valid region path in $\mathcal{A}^{\mathrm{np}}$.

The link between non-punctuality and robustness is as follows, where we abusively denote by $\mathbf{0}$ the zone $\{\mathbf{0}\}$.

Theorem 5.2. *Let $\pi_1\pi_2^\omega$ be a lasso of the timed automaton. We have*

$$\exists \delta > 0 \;\; \mathbf{0} \in \mathsf{CPre}^\delta_{\pi_1}(\nu \mathsf{X}\, \mathsf{CPre}^\delta_{\pi_2}(\mathsf{X})) \iff \mathsf{Post}^{\mathrm{np}}_{\pi_1}(\mathbf{0}) \cap (\textstyle\bigcup_{\delta>0} \nu X\, \mathsf{CPre}^\delta_{\pi_2}(X)) \neq \emptyset$$

Proof. We start by observing that if Z is a zone (with constants over $\mathbb{Q}$), ν is a valuation and e an edge, the three following statements are equivalent: 1. $\exists \delta > 0 \;\; \nu \in \mathsf{CPre}^\delta_e(Z)$; 2. $\nu \in \bigcup_{\delta>0} \mathsf{CPre}^\delta_e(Z)$; 3. $\nu \in \mathsf{Pre}^{\mathrm{np}}_e(Z)$. This is derived from the fact that $\bigcup_{\delta>0} \mathsf{shrink}_{[-\delta,\delta]}([\![g]\!] \cap Z)$ is equal to the intersection of Z with the strict version of g, as present in $\mathcal{A}^{\mathrm{np}}$.

Let us now prove by induction that $\mathsf{Pre}^{\mathrm{np}}_\pi(Z) = \bigcup_{\delta>0} \mathsf{CPre}^\delta_\pi(Z)$ for all paths π. Let us write π equals $\pi' \cdot e$, with e the last edge of π. Then, by inductive hypothesis:

$$\mathsf{Pre}^{\mathrm{np}}_\pi(Z) = \mathsf{Pre}^{\mathrm{np}}_e(\mathsf{Pre}^{\mathrm{np}}_{\pi'}(Z)) = \bigcup_{\delta>0} \mathsf{CPre}^\delta_e(\mathsf{Pre}^{\mathrm{np}}_{\pi'}(Z)) = \bigcup_{\delta>0} \mathsf{CPre}^\delta_e(\bigcup_{\delta'>0} \mathsf{CPre}^{\delta'}_{\pi'}(Z))\,.$$

To finish the induction we need to prove that $\bigcup_{\delta>0} \mathsf{CPre}^\delta_e(\bigcup_{\delta'>0} \mathsf{CPre}^{\delta'}_{\pi'}(Z))$ is equal to $\bigcup_{\delta>0} \mathsf{CPre}^\delta_e(\mathsf{CPre}^\delta_{\pi'}(Z))$. The left to right inclusion holds as for any valuation ν in $\bigcup_{\delta>0} \mathsf{CPre}^\delta_e(\bigcup_{\delta'>0} \mathsf{CPre}^{\delta'}_{\pi'}(Z))$, there exists values for δ and δ' such that $\nu \in \mathsf{CPre}^\delta_e(\mathsf{CPre}^{\delta'}_{\pi'}(Z))$. Moreover, $\mathsf{CPre}^\delta_e(Z)$ and $\mathsf{CPre}^\delta_{\pi'}(Z')$ are decreasing over δ, so that $\nu \in \mathsf{CPre}^{\delta''}_e(\mathsf{CPre}^{\delta''}_{\pi'}(Z))$ with $\delta'' = \min(\delta, \delta')$. For the right to left inclusion, observe that any fixed $\delta > 0$ is covered by the set of all $\delta' > 0$, thus $\mathsf{CPre}^\delta_{\pi'}(Z) \subseteq \bigcup_{\delta'>0} \mathsf{CPre}^{\delta'}_{\pi'}(Z)$, which in turn implies $\mathsf{CPre}^\delta_e(\mathsf{CPre}^\delta_{\pi'}(Z)) \subseteq \mathsf{CPre}^\delta_e(\bigcup_{\delta'>0} \mathsf{CPre}^{\delta'}_{\pi'}(Z))$ as $\mathsf{CPre}^\delta_e(Z')$ is increasing over Z'.

We will also need a duality property on predecessor and successor relations: for all paths π, and zones Z and Z', $Z \cap \mathsf{Pre}^{\mathrm{np}}_\pi(Z') \neq \emptyset$ if and only if $\mathsf{Post}^{\mathrm{np}}_\pi(Z) \cap Z' \neq \emptyset$. This always holds by definition of Pre and Post, in particular in $\mathcal{A}^{\mathrm{np}}$. Then,

$$\begin{aligned} \exists \delta > 0 \;\; \mathbf{0} \in \mathsf{CPre}^\delta_{\pi_1}(\nu X\, \mathsf{CPre}^\delta_{\pi_2}(X)) &\Leftrightarrow \mathbf{0} \in \textstyle\bigcup_{\delta>0} \mathsf{CPre}^\delta_{\pi_1}(\nu X\, \mathsf{CPre}^\delta_{\pi_2}(X)) \\ &\Leftrightarrow \mathbf{0} \cap \mathsf{Pre}^{\mathrm{np}}_{\pi_1}(\nu X\, \mathsf{CPre}^\delta_{\pi_2}(X)) \neq \emptyset \\ &\Leftrightarrow \mathsf{Post}^{\mathrm{np}}_{\pi_1}(\mathbf{0}) \cap (\nu X\, \mathsf{CPre}^\delta_{\pi_2}(X)) \neq \emptyset\,. \end{aligned}$$

□

Therefore, in order to test the robustness of the lasso $\pi_1\pi_2^\omega$, it is enough to only keep in memory the sets $\mathsf{Post}^{\mathrm{np}}_{\pi_1}(\mathbf{0})$ and $\bigcup_{\delta>0} \nu X\, \mathsf{CPre}^\delta_{\pi_2}(X)$.

5.3.2. Forward Analysis

As a consequence of the previous theorem, we can use a classical forward analysis of the timed automaton $\mathcal{A}^{\mathrm{np}}$ to look for the prefix π_1 of the lasso $\pi_1\pi_2^\omega$. A classical inclusion check on zones, as described in Chapter 2.3.5, allows one to stop the exploration, this criterion being complete thanks to Theorem 5.2. It is worth reminding that we consider only bounded clocks, hence the number of reachable zones is finite, ensuring termination.

5.3.3. Robust cycle search

We now perform a second forward analysis, from each possible target location, to find a robust cycle around it. To this end, for each cycle π_2, we must compute the zone $\bigcup_{\delta>0} \nu X\, \mathsf{CPre}^{\delta}_{\pi_2}(X)$. This computation is obtained by arguments developed in Section 5.2.3 (Proposition 5.5). To enumerate cycles π_2, we can again use a classical forward exploration, starting from the universal zone $\top$. Using zone inclusion to stop the exploration is not complete: considering a path π_2' reaching a zone Z_2' included in the zone Z_2 reachable using some π_2, π_2' could be robustly iterable while π_2 is not. In order to ensure termination of our analysis, we instead use reachability relations inclusion checks. These tests are performed using the technique developed in Section 5.1, based on constraint graphs (Theorem 5.1). The correction of this inclusion check is stated in the following lemma, where $\mathsf{Reach}^{\mathrm{np}}(\pi)$ denotes the reachability relation associated with π in the automaton $\mathcal{A}^{\mathrm{np}}$. This result is derived from the analysis based on regions of the previous chapter. Indeed, we will prove that the non-punctual reachability relations we consider capture the existence of non-punctual aperiodic paths in the region automaton, as considered in [SBMR13].

We say that controller wins on the lasso $\pi_0\pi^\omega$ if he wins in the perturbation game restricted to π_0 and π for some δ, *i.e.* if $\exists \delta > 0 \;\; \mathbf{0} \in \mathsf{CPre}^{\delta}_{\pi_0}(\nu \mathsf{X}\, \mathsf{CPre}^{\delta}_{\pi}(\mathsf{X}))$.

Lemma 5.2. *Let π_1 be a path from ℓ_0 to some target location ℓ_t. Let π_2, π_2' be two paths from ℓ_t to some location ℓ, such that $\mathsf{Reach}^{\mathrm{np}}(\pi_2) \subseteq \mathsf{Reach}^{\mathrm{np}}(\pi_2')$. For all paths π_3 from ℓ to ℓ_t, if controller wins on the lasso $\pi_1(\pi_2\pi_3)^\omega$, then controller wins on the lasso $\pi_1(\pi_2'\pi_3)^\omega$*

This lemma proved to be a major technical challenge, and the rest of this section is dedicated to its proof.

As explained earlier, one can assume without loss of generality that all constraints in the guards of $\mathcal{A}$ are strict. We will thus make no distinction in this proof between Post and $\mathsf{Post}^{\mathrm{np}}$, or Reach and $\mathsf{Reach}^{\mathrm{np}}$.

In order to prove Lemma 5.2, we will need new notions on regions. Recall that a region r is defined by a valuation ι with integer coordinates and a partition $R_0, \ldots, R_m$ of $\mathcal{X}$, such that a valuation ν belongs to r when for every clock $\mathsf{x} \in \mathcal{X}$, the integral part of $\nu(\mathsf{x})$ is equal to $\iota(\mathsf{x})$, and the fractional part of the coordinates of ν satisfies the ordering $0 = R_0 < R_1 < R_2 < \cdots < R_m$ (*i.e.* for all clocks x in R_0, $\mathrm{frac}(\nu(\mathsf{x})) = 0$, for all $0 \leqslant i \leqslant m$ and all clocks $\mathsf{x}, \mathsf{y} \in R_i$, $\mathrm{frac}(\nu(\mathsf{x})) = \mathrm{frac}(\nu(\mathsf{y}))$, and for all $0 \leqslant i < j \leqslant m$ and all clocks $\mathsf{x} \in R_i$, $\mathsf{y} \in R_j$, $\mathrm{frac}(\nu(\mathsf{x})) < \mathrm{frac}(\nu(\mathsf{y}))$). Here we abusively write $0 = R_0$ to remind that the clocks in R_0 have integer values. Moreover, the set R_0 can be empty but not the sets $R_1, \ldots, R_m$. Region r is non-punctual if and only if $R_0 = \emptyset$.

Consider a region r with corresponding integral part ι and clock ordering $0 = R_0 < \cdots < R_m$. The first time-successor of r is the region defined by ι and $0 < R_0 < R_1 \cdots < R_m$ if $R_0 \neq \emptyset$, by ι' and $0 = R_m < R_1 \cdots < R_{m-1}$ if $R_0 = \emptyset$, with $\iota'(\mathsf{x}) = \iota(\mathsf{x}) + 1$ if $\mathsf{x} \in R_m$ and $\iota'(\mathsf{x}) = \iota(\mathsf{x})$ otherwise. Intuitively, it is the next region that valuations in r reach by letting time elapse. The i-th time successor of r is defined inductively with the first time-successor notion, with the 0-th time successor of r set to r. Any region reachable from r by time-elapse can be assigned an $i \geqslant 0$ such that it is the i-th time successor of r.

Symmetrically, the first time-predecessor of r is the region defined by ι and $0 = R_1 < \cdots < R_m$ if $R_0 = \emptyset$, by ι' and $0 < R_1 \cdots < R_m < R_0$ if $R_0 \neq \emptyset$, with $\iota'(\mathsf{x}) = \iota(\mathsf{x}) - 1$ if $\mathsf{x} \in R_0$ and $\iota'(\mathsf{x}) = \iota(\mathsf{x})$ otherwise.

We introduce a notion of *first partial-time-predecessor* of a region r, characterized by a subset P_0 of R_0. Intuitively, it performs the first time-predecessor operation on all clocks except $R_0 \backslash P_0$.

Definition 5.1. The *first partial-time-predecessor* of a region r, characterized by a subset $P_0 \neq \emptyset$ of R_0, is the region defined by ι' and $0 = (R_0 \backslash P_0) < R_1 \cdots < R_m < P_0$, with $\iota'(\mathsf{x}) = \iota(\mathsf{x}) - 1$ if $\mathsf{x} \in P_0$ and $\iota'(\mathsf{x}) = \iota(\mathsf{x})$ otherwise. Moreover, let the first partial-time-predecessor of r characterized by $\emptyset$ be r.

If r is punctual (*i.e.* not non-punctual), the set of all first partial-time-predecessors of r contains r (characterised by $P_0 = \emptyset$) and the first time-predecessor of r (characterised by $P_0 = R_0$). If r is non-punctual, the set of all first partial-time-predecessors of r is $\{r\}$.

We introduce a way to refine a clock partition. Intuitively, a clock ordering $0 = R'_0 < \cdots < R'_{m'}$ is an ordered sub-partition of $0 = R_0 < \cdots < R_m$ if some sets R_i are split into several $R'_{i'} < \cdots < R'_{j'}$.

Definition 5.2. $0 = R'_0 < \cdots < R'_{m'}$ is an *ordered sub-partition* of $0 = R_0 < \cdots < R_m$ if $R'_0 \subseteq R_0$ and for all $0 \leqslant i < j \leqslant m$, $\mathsf{x} \in R_i$ and $\mathsf{y} \in R_j$, there exists $0 \leqslant i' < j' \leqslant m'$ such that $\mathsf{x} \in R'_{i'}$ and $\mathsf{y} \in R'_{j'}$.

Consider two regions r and r', with corresponding integral parts ι and ι', and clock orderings $0 = R_0 < \cdots < R_m$ and $0 = R'_0 < \cdots < R'_{m'}$.

Definition 5.3. We say that r *is simulated by* r', and write $r \preccurlyeq r'$, if there exists r'', a first partial-time-predecessor of r with corresponding integral part ι'' and clock ordering $0 = R''_0 < \cdots < R''_m$, such that $\iota' = \iota''$ and $0 = R'_0 < \cdots < R'_{m'}$ is an ordered sub-partition of $0 = R''_0 < \cdots < R''_m$.

Definition 5.4. We will say that r and r' are *equivalent w.r.t.* $\mathcal{Y}$ with $\mathcal{Y} \subseteq \mathcal{X}$ if they are equal when projected on $\mathbb{R}^{\mathcal{Y}}_{\geqslant 0}$. Equivalently, if there exist two valuations $\nu \in r$ and $\nu' \in r'$ such that for all $\mathsf{x} \in \mathcal{Y}$, $\nu(\mathsf{x}) = \nu'(\mathsf{x})$.

Observe that if r and r' are equivalent w.r.t. $\mathcal{Y}$, then $r[R := 0] = r'[R := 0]$ for all resets R with $\mathcal{X} \backslash \mathcal{Y} \subseteq R$. In particular, if two regions r and r' are equivalent w.r.t. $\mathcal{X}$ then $r = r'$. All regions r and r' are equivalent w.r.t. $\emptyset$.

Example 5.1. Figure 5.4 represents the simulation and equivalence notions on a few examples. On the left, the blue region characterised by $\iota = (0, 0)$ and $0 < \{\mathsf{x}_1, \mathsf{x}_2\}$ is simulated by the two light blue regions, because $0 < \{\mathsf{x}_1\} < \{\mathsf{x}_2\}$ and $0 < \{\mathsf{x}_2\} < \{\mathsf{x}_1\}$ are ordered sub-partitions of $0 < \{\mathsf{x}_1, \mathsf{x}_2\}$. The red region characterised by $\iota = (1, 1)$ and $0 = \{\mathsf{x}_1\} < \{\mathsf{x}_2\}$ is simulated by the two light red regions, because $0 < \{\mathsf{x}_1\} < \{\mathsf{x}_2\}$ is an ordered sub-partition of $0 = \{\mathsf{x}_1\} < \{\mathsf{x}_2\}$, and $\iota = (0, 1)$ and $0 < \{\mathsf{x}_2\} < \{\mathsf{x}_1\}$ is a first partial-time predecessor of the red region. The green region is only simulated by itself. The green and blue regions are equivalent w.r.t. $\{\mathsf{x}_2\}$. On the right, Figure 5.4 represents

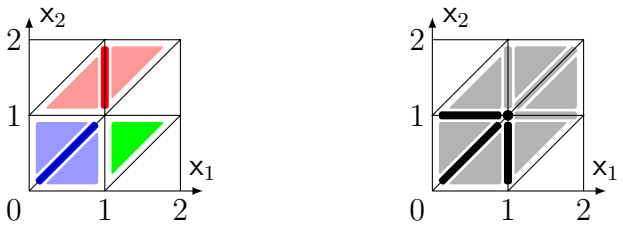


Figure 5.4.: The simulation relation between regions in $\mathsf{Reg}(\{\mathsf{x}_1, \mathsf{x}_2\}, 2)$.

the set of regions simulated by the region characterised by $\iota = (1, 1)$ and $0 = \{\mathsf{x}_1, \mathsf{x}_2\}$. The black regions are its first partial-time predecessors, and the gray regions are obtained by ordered sub-partition.

The relation $\preccurlyeq$ enjoys nice properties, that we now describe.

Lemma 5.3. *For all regions r and r', with corresponding integral parts ι and ι', and clock orderings $0 = R_0 < \cdots < R_m$ and $0 = R'_0 < \cdots < R'_{m'}$, such that $r \preccurlyeq r'$ and r, r' are equivalent w.r.t. $\mathcal{Y} \subseteq \mathcal{X}$:*

1. *if r is non-punctual then r' is non-punctual as well;*
2. *if r is punctual and s is the first time-successor of r, s is non-punctual and $r \preccurlyeq s$;*
3. *every strict non-diagonal guard satisfied by r is also satisfied by r';*
4. *for each reset $R \subseteq \mathcal{X}$, $r[R := 0] \preccurlyeq r'[R := 0]$ and $r[R := 0]$, $r'[R := 0]$ are equivalent w.r.t. $\mathcal{Y} \cup R$;*
5. *for each time-successor s of r there exists a time-successor s' of r' such that $s \preccurlyeq s'$ and s, s' are equivalent w.r.t. $\mathcal{Y}$;*
6. *if $r' \preccurlyeq s$ for some region s then $r \preccurlyeq s$.*

Proof. Let r'' be a first partial-time-predecessor of r characterised by $P_0 \subseteq R_0$, with corresponding integral part $\iota'' = \iota'$ and clock ordering $0 = R''_0 < \cdots < R''_m$, such that $0 = R'_0 < \cdots < R'_{m'}$ is an ordered sub-partition of $0 = R''_0 < \cdots < R''_m$.

1. If $R_0 = \emptyset$ then $R'_0 \subseteq R''_0 \subseteq R_0 = \emptyset$, and r' is non-punctual.
2. The first time-successor of r is the non-punctual region s defined by ι and $0 < R_0 < R_1 \cdots < R_m$ since r is punctual. It is an ordered sub-partition of r which is a first partial-time-predecessor of itself, therefore $r \preccurlyeq s$.
3. We show that a strict non-diagonal atomic constraint satisfied by r is also satisfied by r''. If r is non-punctual then $r'' = r$, and they satisfy the same constraints. If r is punctual, a constraint $\mathsf{x} < c$ with $\mathsf{x} \in \mathcal{X}$ is satisfied by r if and only if $c \geqslant \iota(\mathsf{x}) + 1$. Since $\iota''(\mathsf{x}) \leqslant \iota(\mathsf{x})$ for all $\mathsf{x} \in \mathcal{X}$, $c \geqslant \iota''(\mathsf{x}) + 1$ and r'' satisfies $\mathsf{x} < c$. A constraint

$\mathsf{x} > c$ with $\mathsf{x} \in \mathcal{X}$ is satisfied by r if and only if $c < \iota(\mathsf{x})$ or $\mathsf{x} \in \mathcal{X} \backslash R_0 \wedge c = \iota(\mathsf{x})$. If $c < \iota(\mathsf{x})$, then either $c < \iota''(\mathsf{x})$ or $c = \iota''(\mathsf{x})$ and $\mathsf{x} \in P_0$, and in both cases $\mathsf{x} > c$ is satisfied by r''. If $\mathsf{x} \in \mathcal{X} \backslash R_0 \wedge c = \iota(\mathsf{x})$, then $\mathsf{x} \notin (R_0 \backslash P_0) \wedge c = \iota''(\mathsf{x})$ and $\mathsf{x} > c$ is satisfied by r''.

Then, as $\iota' = \iota''$ and R'_0 is included in R''_0, all strict non-diagonal constraints satisfied by r'' are also satisfied by r'.

4. The simulation relation is stable by projection into $\mathbb{R}_{\geqslant 0}^{\mathcal{X} \backslash R}$, and it is also stable by adding new clocks to both R_0 and R'_0, therefore it is stable by reset. Equivalence w.r.t. $\mathcal{Y}$ is maintained and extended to $\mathcal{Y} \cup R$ by definition.

5. We prove by induction on i that for all $i \geqslant 0$, there exists $j \geqslant 0$ such that the i-th time-successor of r is simulated by the j-th time-successor of r' and that they are equivalent w.r.t. $\mathcal{Y}$. If $i = 0$, pick $j = 0$. For the induction step, it is enough to prove the property with $i = 1$. Let us first assume that r is non-punctual. The first time-successor of r is described by $\iota'$and $0 = R_m < R_1 < \cdots < R_{m-1}$, with ι' equal to $\iota + 1$ on R_m and equal to ι on all other clocks. Moreover, r' is described by ι and an ordered sub-partition of $0 < R_1 < \cdots < R_m$. Let us assume that R_m is split into $R'_{m'-j} < \cdots < R'_{m'}$ in r', with $j \geqslant 0$. Therefore, the $(2j+1)$-th time-successor of r' is described by ι' and an ordered sub-partition of the clock ordering $0 = R_m < R_1 < \cdots < R_{m-1}$, and therefore simulates the first time-successor of r. Additionally, r and r' induce the same ordering for clocks in $\mathcal{Y}$, and this is maintained on the successors, so they are equivalent w.r.t. $\mathcal{Y}$. Let us now assume that r is punctual. The first time-successor of r is described by ι and $0 < R_0 < R_1 < \cdots < R_m$. Moreover, r' is described by ι' and an ordered sub-partition of $0 = (R_0 \backslash P_0) < R_1 \cdots < R_m < P_0$, with $\iota'(\mathsf{x}) = \iota(\mathsf{x}) - 1$ if $\mathsf{x} \in P_0$ and $\iota'(\mathsf{x}) = \iota(\mathsf{x})$ otherwise. Let us assume that P_0 is split into $R'_{m'-j} < \cdots < R'_{m'}$ in r', with $j \geqslant 0$. Then, the $(2j+1)$-th time-successor of r' is described by ι and an ordered sub-partition of the clock ordering $0 = P_0 < (R_0 \backslash P_0) < R_1 < \cdots < R_m$, which is an ordered sub-partition of the clock ordering $0 = R_0 < R_1 < \cdots < R_m$, and therefore simulates the first time-successor of r. Additionally, r and r' induce the same ordering for clocks in $\mathcal{Y}$, and this is maintained on the successors, so they are equivalent w.r.t. $\mathcal{Y}$.

6. Let s correspond to an integral part ι^s and a clock ordering $0 = R_0^s < \cdots < R_{m^s}^s$. Let s' be a first partial-time-predecessor of r' characterised by $Q_0 \subseteq R'_0$ with corresponding integral part $\iota^{s'} = \iota'$ and clock ordering $0 = R_0^{s'} < \cdots < R_{m'}^{s'}$, such that $0 = R_0^s < \cdots < R_{m^s}^s$ is an ordered sub-partition of $0 = R_0^{s'} < \cdots < R_{m'}^{s'}$. Now, if r' is non-punctual then $r' = s'$. As being an ordered sub-partition is transitive, $0 = R_0^s < \cdots < R_{m^s}^s$ is an ordered sub-partition of $0 = R''_0 < \cdots < R''_m$, and $r \preccurlyeq s$. If r' is punctual, the region r must be punctual by Lemma 5.3.1, and $Q_0 \subseteq R_0 \backslash P_0$. Then, the clock ordering of s is an ordered sub-partition of the clock ordering of the first partial-time-predecessor of r characterised by $P_0 \uplus Q_0$ (the last sets R_i^s in the clock ordering of r^s all belong to $P_0 \uplus Q_0$), thus $r \preccurlyeq s$.

□

Consider a region path p describing a sequence

$$r_1 \xrightarrow{\text{delay}} r_2 \xrightarrow{g_1,\mathcal{Y}_1} r_3 \xrightarrow{\text{delay}} \dots \xrightarrow{g_{|\pi|},\mathcal{Y}_{|\pi|}} r_n\,,$$

and p' another region path $r'_1 \xrightarrow{\text{delay}} \dots \xrightarrow{g_{|\pi|},\mathcal{Y}_{|\pi|}} r'_n$ following the same sequence of edges π. We say that $\mathsf{p} \preccurlyeq \mathsf{p}'$ if for all $1 \leqslant i \leqslant n$, $r_i \preccurlyeq r'_i$. In particular, if $\mathsf{p} \preccurlyeq \mathsf{p}'$ and p is non-punctual (*i.e.* ever region r_{2i+1} reached after a delay is non-punctual), p' is non-punctual as well by Lemma 5.3.1.

The following results can be obtained from Lemma 5.3, and state that $\preccurlyeq$ acts as a time-abstract simulation relation, while keeping additional guarantees on progress cycles or non-punctual region paths.

Lemma 5.4. *Let π be a path in $\mathcal{A}$ describing a progress cycle, let p be a region path from a region r_1 to a region r_2 that follows π, and let r'_1 be a region such that $r_1 \preccurlyeq r'_1$. Then, there exists a region path p' from r'_1 to r_2 following π, such that $\mathsf{p} \preccurlyeq \mathsf{p}'$.*

Proof. Consider an edge in $\mathcal{A}$ of guard g and reset R, and a region path $r_1 \xrightarrow{\text{delay}} r_2 \xrightarrow{g,R} r_3$ and let r'_1 be a region such that r_1 and r'_1 are equivalent w.r.t. $\mathcal{Y}$ with some $\mathcal{Y} \subseteq \mathcal{X}$, and $r_1 \preccurlyeq r'_1$. Then, by Lemma 5.3.5, there exists a region r'_2 time-successor of r'_1 such that $r_2 \preccurlyeq r'_2$ and r_2, r'_2 are equivalent w.r.t. $\mathcal{Y}$. Since $r_2 \models g$, by Lemma 5.3.3, $r'_2 \models g$. Finally, if $r'_3 = r'_2[R := 0]$ and $\mathsf{p}' = r'_1 \xrightarrow{\text{delay}} r'_2 \xrightarrow{g,R} r'_3$, by Lemma 5.3.4 we obtain $r_3 \preccurlyeq r'_3$ and r_3, r'_3 are equivalent w.r.t. $\mathcal{Y} \cup R$. By applying this procedure inductively on p (starting from r_1 and r'_1 equivalent w.r.t. $\emptyset$), we obtain a region path p' from r'_1 to some r'_2 following π, such that $\mathsf{p} \preccurlyeq \mathsf{p}'$ and r_2, r'_2 are equivalent w.r.t. $\bigcup_{(\ell,g,R,\ell')\in\pi} R$ the clocks reset by π. As π is a progress cycle, it resets all clocks, and therefore $r'_2 = r_2$. □

Lemma 5.5. *Let π be a path in $\mathcal{A}$, and let p be a region path following π from a region r_1 to a region r_2. Then, there exists a non-punctual region path p' following π from r_1 to some r'_2 with $\mathsf{p} \preccurlyeq \mathsf{p}'$.*

Proof. Consider an edge in $\mathcal{A}$ of guard g and reset R, and a region path $r_1 \xrightarrow{\text{delay}} r_2 \xrightarrow{g,R} r_3$, and let r'_1 be a region such that $r_1 \preccurlyeq r'_1$. Then, by Lemma 5.3.5, there exists a region r''_2 time-successor of r'_1 such that $r_2 \preccurlyeq r''_2$. If r''_2 is non-punctual, let $r'_2 = r''_2$, otherwise let r'_2 be the first time-successor of r''_2. By Lemma 5.3.2 and Lemma 5.3.6, r'_2 is a time-successor of r'_1 such that $r_2 \preccurlyeq r'_2$. Since $r_2 \models g$, by Lemma 5.3.3, $r'_2 \models g$. Finally, if $r'_3 = r'_2[R := 0]$ and $\mathsf{p}' = r'_1 \xrightarrow{\text{delay}} r'_2 \xrightarrow{g,R} r'_3$, by Lemma 5.3.4 we obtain $r_3 \preccurlyeq r'_3$. By applying this procedure inductively on p, we obtain a non-punctual region path p' from r'_1 to some r'_2 following π, with $\mathsf{p} \preccurlyeq \mathsf{p}'$. □

We will now show that region cycles can be made non-punctual while keeping a full reachability relation on the associated path in $\mathcal{A}$.

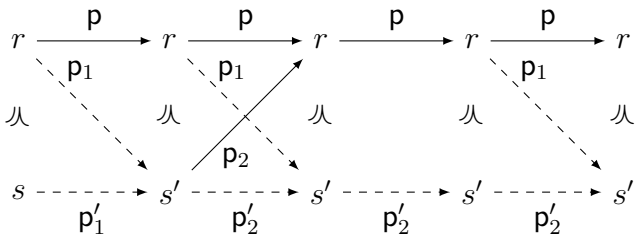


Figure 5.5.: Proof scheme of Lemma 5.6. Edges represent region paths, the dashed ones are non-punctual.

Lemma 5.6. *Let π be a path in $\mathcal{A}$ describing a progress cycle, Let r be a region such that $r \times r \subseteq \mathsf{Reach}(\pi)$. Let p be a region cycle around region r that follows π, and let s be a region such that $r \preccurlyeq s$. There exists a region s' reachable from s by a non-punctual region path p'_0 that follows π, and a non-punctual region cycle p' around s' that follows π^3, with $s' \times s' \subseteq \mathsf{Reach}(\pi^3)$.*

Proof. Let us explain how to construct a region s' and four region paths p_1, p_2, p'_1 and p'_2 following π such that

- $\mathsf{p} \preccurlyeq \mathsf{p}_1$,
- $\mathsf{p} \preccurlyeq \mathsf{p}_2$,
- $\mathsf{p}_1 \preccurlyeq \mathsf{p}'_1$,
- $\mathsf{p}_1 \preccurlyeq \mathsf{p}'_2$,
- p_1 is a non-punctual path from r to s',
- p_2 is a path from s' to r,
- p'_1 is a non-punctual paths from s to s',
- and p'_2 is a non-punctual paths from s' to s'.

We start by applying Lemma 5.5 on p, and obtain a non-punctual path p_1 from r to a region s' with $\mathsf{p} \preccurlyeq \mathsf{p}_1$. Now, we apply Lemma 5.4 on p_1 and s (as $r \preccurlyeq s$), and obtain a path p'_1 from s to s' with $\mathsf{p}_1 \preccurlyeq \mathsf{p}'_1$ (and therefore p'_1 is also non-punctual). Then we apply Lemma 5.4 on p_1 and s' (as $r \preccurlyeq s'$), and obtain a path p'_2 from s' to s' with $\mathsf{p}_1 \preccurlyeq \mathsf{p}'_2$ (and therefore p'_2 is also non-punctual). Finally, by applying Lemma 5.4 on r, s' and p (as $r \preccurlyeq s'$), we obtain a region path p_2 that follows π, starts in s' and ends in r. We fix $\mathsf{p}'_0 = \mathsf{p}'_1$ and $\mathsf{p}' = (\mathsf{p}'_2)^3$. Let us show $s' \times s' \subseteq \mathsf{Reach}(\pi^3)$. Let ν, ν' be two valuations in s'. There exists a region path p_2 that follows π, starts in s' and ends in r. Thus, ν can reach some valuation ν_1 in r by following π. We also know that p_1 is a region path from r to s' following π, therefore there exists a valuation ν'_1 in r that can reach ν' by following π. From $r \times r \subseteq \mathsf{Reach}(\pi)$, we deduce that ν_1 can reach ν'_1 by following π. Therefore every valuation ν in s' can reach every valuation ν' in s' by following π^3.

□

We are now ready to prove Lemma 5.2.

Proof of Lemma 5.2. Let π_1 a path from ℓ_0 to some target location ℓ_t. Let π_2, π_2' be two paths from ℓ_t to some location ℓ, such that $\mathsf{Reach}(\pi_2) \subseteq \mathsf{Reach}(\pi_2')$. Let us assume that controller wins on the lasso $\pi_1(\pi_2\pi_3)^\omega$. Then, the controller wins in the perturbation game restricted to π_1, π_2 and π_3 and with ℓ_t as only target, for some δ. Therefore, there exists non-punctual region paths p_4 and p_5 in the region abstraction of $\mathcal{A}$, such that p_4 reaches a target region (ℓ_t, r) from $(\ell_0, \mathbf{0})$, and p_5 is an aperiodic cycle around r. There exists k such that the reachability relation of p_5^k is complete. Thus, we can define π_4 a path of the form $\pi_1(\pi_2\pi_3)^i$ with $i \geqslant 0$, and π_5 a cycle of the form $(\pi_2\pi_3)^j$ with $j > 0$, such that $r \times r \subseteq \mathsf{Reach}(\pi_5)$.

Consider $\pi_4' = \pi_1(\pi_2'\pi_3)^i$ and $\pi_5' = (\pi_2'\pi_3)^j$. From $\mathsf{Reach}(\pi_2) \subseteq \mathsf{Reach}(\pi_2')$, we derive $\mathsf{Reach}(\pi_2\pi_3) \subseteq \mathsf{Reach}(\pi_2'\pi_3)$ and thus $\mathsf{Reach}(\pi_4) \subseteq \mathsf{Reach}(\pi_4')$ and $\mathsf{Reach}(\pi_5) \subseteq \mathsf{Reach}(\pi_5')$. Therefore, $r \times r \subseteq \mathsf{Reach}(\pi_5')$, $\mathsf{Post}_{\pi_4}(\mathbf{0}) \subseteq \mathsf{Post}_{\pi_4'}(\mathbf{0})$ and $\mathsf{Post}_{\pi_5}(r) \subseteq \mathsf{Post}_{\pi_5'}(r)$, and we can define a region path p_4' from $\mathbf{0}$ to r along π_4', and a region path p_5' from r to r along π_5'. Note that $r \times r \subseteq \mathsf{Reach}(\pi_5')$ implies that π_5' is a progress cycle, as otherwise no execution following π_5' can start and end in the same valuation.

By using Lemma 5.5 on π_4' and p_4', we can obtain a non-punctual region path p_4'' from $\mathbf{0}$ to some region r_0 that follows π_4', such that $r \preccurlyeq r_0$. Now, we can apply Lemma 5.6 on π_5', r and r_0, and obtain a non-punctual lasso following $\pi_5'\pi_5'^3$, starting from r_0 and cycling around r_1, such that $r_1 \times r_1 \subseteq \mathsf{Reach}(\pi_5'^3)$. Therefore, by Lemma 4.2, controller wins in the perturbation game restricted to π_1, π_2' and π_3 for some δ, and thus wins on the lasso $\pi_1(\pi_2'\pi_3)^\omega$. □

5.4. Case study

We implemented our algorithm in C++. We rely on the model-checking tool TChecker [HPT19] for efficiently exploring the zone abstraction, and implemented an on-the-fly construction of reachability relations using DBMs, and a check for the robust iterability of a cycle based on shrunk DBMs.

To illustrate our approach, we present a case study on the regulation of train networks. Urban train networks in big cities are often particularly busy during rush hours: trains run in high frequency so even small delays due to incidents or passenger misbehaviour can perturb the traffic and end up causing large delays. Train companies thus apply regulation techniques: they slow down or accelerate trains, and modify waiting times in order to make sure that the traffic is fluid along the network. Computing robust schedules with provable guarantees is a difficult problem (see *e.g.* [DPH07]).

We study here a simplified model of a train network and aim at automatically synthesizing a controller that regulates the network despite perturbations, in order to ensure performance measures on total travel time for each train. Consider a circular train network with m stations $s_0, \ldots, s_{m-1}$ and n trains. We require that all trains are at distinct stations at all times. For each station $i \in [0, m-1]$, there is an attached interval of delays $[u_i, v_i]$ which bounds the travel time from s_i to $s_{i+1 \bmod m}$. Here the lower bound

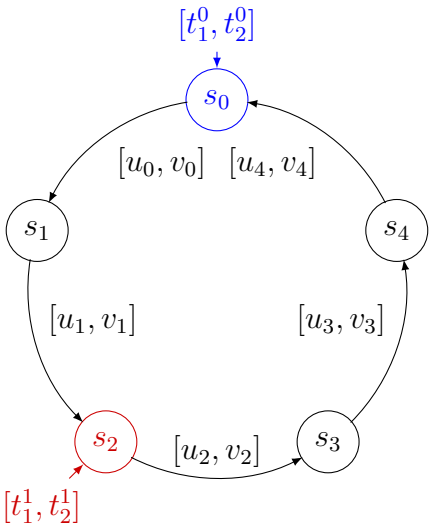


Figure 5.6.: A simple circular train network, with five stations and two trains. The blue (resp. red) train starts at station s_0 (resp. s_2) and must cycle the network within time $[t_1^0, t_2^0]$ (resp. $[t_1^1, t_2^1]$).

comes from physical limits (maximal allowed speed, and travel distance) while the upper bound comes from operator specification (*e.g.* it is not desirable for a train to remain at station for more than 3 minutes). The objective of each train $j \in [0, n-1]$ is to cycle on the network while completing each tour within a given time interval $[t_1^j, t_2^j]$. An example for $m = 5$ and $n = 2$ is displayed in Figure 5.6.

All timing requirements are naturally encoded with clocks: every train $i \in [0, n-1]$ is associated two clocks x^i, y^i, and a timed automaton $\mathcal{A}^i$ defined as in Figure 5.7. The final timed automaton $\mathcal{A}$ is obtained as the product automaton $\mathcal{A}^0 \times \cdots \times \mathcal{A}^{n-1}$, whose locations are tuples $(s^0, \ldots, s^{n-1})$ associating a station for every train, and whose edges are all

$$(s^0, \ldots, s^i, \ldots, s^{n-1}) \xrightarrow{g^i, \mathcal{Y}^i} (s^0, \ldots, q^i, \ldots, s^{n-1}),$$

with $s^i \xrightarrow{g^i, \mathcal{Y}^i} q^i$ an edge of $\mathcal{A}^i$. Locations such that two trains are positioned at the same station are finally removed.

Given the model $\mathcal{A}$, we solve the qualitative robust controller synthesis problem in order to find a controller choosing travel times for all trains ensuring a Büchi condition (visiting s_0 infinitely often). Given the fact that trains cannot be at the same station simultaneously, it suffices to state the Büchi condition only for one train, since its satisfaction of the condition necessarily implies that of all other trains.

Let us present two representative instances and then comment the performance of the algorithm on a set of instances. Consider a network with two trains and m stations, with $[u_i, v_i] = [200, 400]$ for each station i, and the objective of both trains is the interval $[250 \cdot m, 350 \cdot m]$, that is, an average travel time between stations that lies in $[250, 350]$. The algorithm finds an accepting lasso: intuitively, by choosing δ small enough

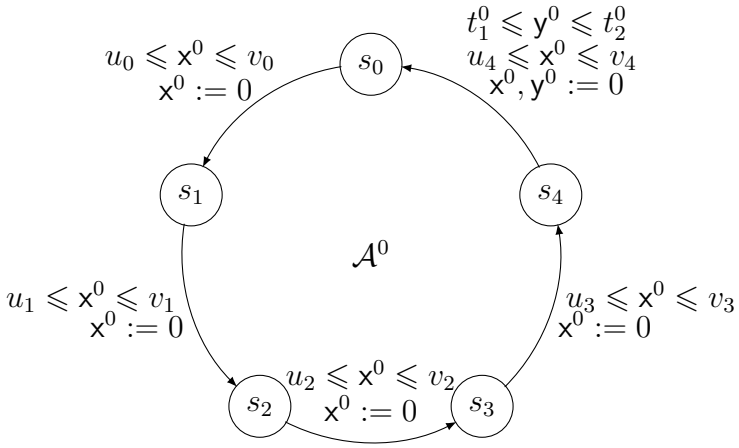


Figure 5.7.: The timed automaton encoding the blue train of Figure 5.6.

Scenario	m	n	#Clocks	robust?	time
A	6	2	4	yes	4s
B	6	2	4	no	2s
C	6	3	5	no	263s
D	6	3	4	yes	125s
E	6	4	2	yes	53s
F	6	4	2	yes	424s
G	6	4	8		TO
H	6	4	8		TO
I	20	2	2	yes	76s
J	20	2	2	yes	55s
K	30	2	2	yes	579s

Table 5.1.: Summary of experiments with different sizes. In each scenario, we assign a different objective to a subset of trains. The answer is *yes* if a robust controller was found, *no* if none exists. TO stands for a time-out of 30 minutes.

so that $m\delta < 50$, perturbations do not accumulate too much and the controller can always choose delays for both trains and satisfy the constraints. This case corresponds to scenario A in Table 5.1. Consider now the same network but with two different objectives: $[0, 300 \cdot m]$ and $[300 \cdot m, \infty)$. Thus, one train needs to complete each cycle in at most $300 \cdot m$ time units, while the other one in at least $300 \cdot m$ time units. A classical Büchi emptiness check reveals the existence of an accepting lasso: it suffices to move each train in exactly 300 time units between each station. This controller can even recover from perturbations for a bounded number of cycles: for instance, if a train arrives late at a station, the next travel time can be chosen smaller than 300. However, such corrections will cause the distance between the two trains to decrease and if such perturbations happen regularly, the system will eventually enter a deadlock. Our algorithm detects that there is no robust controller for the Büchi objective. This corresponds to the scenario B in Table 5.1. Other scenarios are defined similarly for varying number of stations and trains, and sometimes some trains are not given an objective $[t_1^i, t_2^j]$ in order to reduce the number of clocks.

Table 5.1 summarizes the outcome of our prototype implementation on other scenarios. The tool was run on a 3.2Ghz Intel i7 processor running Linux, with a 30 minute time out and 2GB of memory. The performance is sensitive to the number of clocks: on scenarios with 8 clocks the algorithm ran out of time.

6. The quantitative problem

We say that a perturbation δ is *admissible* if the controller wins the game $\mathcal{G}_\delta(\mathcal{A})$. The qualitative robust controller synthesis problem, solved in the previous chapters, aims at deciding whether *there exists* a positive admissible perturbation. The more ambitious quantitative problem consists in determining the *largest admissible* perturbation.

The algorithm developed in Chapter 5 relied on solving the qualitative problem on lassos by performing a bounded $(2|\mathcal{X}_0|^2)$ number of computations of the CPre^δ_π operator. For the quantitative problem, instead of focusing on arbitrarily small values of δ using shrunk DBMs as we did previously, we must perform a computation that holds for all values of δ.

To do so, we consider an extension of the (shrunk) DBMs in which each entry of the matrix (which thus represents a clock constraint) is a piecewise affine function of δ. We argue that all the operations involved in the computation of the CPre^δ_π operator can be performed symbolically w.r.t. δ using such functions.

6.1. Parametric DBMs

Formally, a parametric bound is a mapping f from $\mathbb{R}_{\geqslant 0}$ to $\mathsf{Bounds}(\mathbb{R})$, such that $f(\delta)$ represents the bound associated to the maximal perturbation δ. We define the operations min, inf and $+$ over parametric bounds in an intuitive way, with $\min(f,g) : \delta \mapsto \min(f(\delta), g(\delta))$, $\inf\{f_1, f_2, \ldots\} : \delta \mapsto \inf\{f_1(\delta), f_2(\delta), \ldots\}$ and $f + g : \delta \mapsto f(\delta) + g(\delta)$, by using the min, inf and $+$ operations of $\mathsf{Bounds}(\mathbb{R})$.

Lemma 6.1. *Parametric bounds equipped with* min *and* $+$ *form an ordered semiring with closure. Moreover, the closure* $f^{(*)}$ *of a parametric bound* f *is equal to* $\delta \mapsto (f(\delta))^{(*)}$.

Proof. As $(\mathsf{Bounds}(\mathbb{R}), \min, +)$ is a semiring, one can derive that parametric bounds form a semiring with min and $+$. Consider the relation $\sqsubseteq$ defined over parametric bounds as $\{(f,g) \mid \exists h, \min(f,h) = g\}$. Observe that $f \sqsubseteq g$ if and only if $\min(f,g) = g$. Therefore, $\sqsubseteq$ is antisymmetric. It follows that the semiring of parametric bounds is ordered. A difference with the previous cases of standard or shrunk DBMs is that the min operator is not selective and thus the order it induces is not total. Moreover, $\sqsubseteq$ is complete as the set of parametric bounds forms a complete join semilattice with $\sqsubseteq$, where join is the inf operator. Therefore, the semiring of parametric bounds has closure. The closure $f^{(*)}$ of a parametric bound f equals

$$\inf\{(\delta \mapsto (\leqslant, 0)), f, f + f, f + f + f, \ldots\},$$

and thus maps $\delta \in \mathbb{R}_{\geqslant 0}$ to $(f(\delta))^{(*)}$ by definition of inf. □

6.1.1. Piecewise affine bounds

We say that a parametric bound f is a finite affine bound if there exists $\prec \in \{<, \leqslant\}$, $c \in \mathbb{Z}$ and $p \in \mathbb{N}$ such that for all $\delta \in \mathbb{R}_{\geqslant 0}$, $f(\delta) = (\prec, c - \delta p)$. The equation of f refers to the expression $\prec c - \delta p$, where c is called the constant and p is called the slope. We also define infinite affine bounds as equal to either $\delta \mapsto (<, +\infty)$ or $\delta \mapsto (<, -\infty)$. Then, we define the class of *piecewise affine bounds* PWBounds as the smallest set of parametric bounds that contains the affine bounds, is stable by min and $+$, and is stable by the closure operation $f \mapsto f^{(*)}$, *i.e.* for every parametric bounds $f, g \in$ PWBounds, $f + g$, $\min(f, g)$ and $f^{(*)}$ are in PWBounds. Some examples are displayed in Figure 6.1.

Such a parametric bound f is a piecewise affine mapping with finitely many pieces. Note that all affine bounds are non-increasing, *i.e.* if $\delta \leqslant \delta'$ then $f(\delta') \preccurlyeq f(\delta)$, and that the operations min, $+$ and closure map non-increasing parametric bounds to non-increasing parametric bounds, such that all bounds in PWBounds are non-increasing. If f contains finite pieces only, *i.e.* it maps all δ to finite bounds, it can be described as a finite sequence

$$(\delta_0, f(0)) \xrightarrow{\prec_1 c_1 - \delta p_1} (\delta_1, f(\delta_1)) \xrightarrow{\prec_2 c_2 - \delta p_2} \dots (\delta_k, f(\delta_k)) \xrightarrow{\prec_{k+1} c_{k+1} - \delta p_{k+1}} \infty$$

with $k \in \mathbb{N}$, $\delta_0 = 0$ and $\delta_i \in \mathbb{R}_{\geqslant 0}$ for $i \in [1, k]$, such that for all $i \in [1, k], \delta \in (\delta_{i-1}, \delta_i)$, it holds that $f(\delta) = (\prec_i, c_i - \delta p_i)$, and for all $\delta > \delta_k$, $f(\delta) = (\prec_{k+1}, c_{k+1} - \delta p_{k+1})$. The number of pieces of f is $k + 1$, and the values δ_i with $i \in [0, k]$ are called *split points*, and mark the junction between pieces. We say that f contains an initial split point at $\delta = 0$ and k finite-finite split points for values of $\delta > 0$.

If f contains some infinite pieces, then either f maps every δ to $(<, +\infty)$, or it contains a sequence of finite pieces and a last piece that maps every δ to $(<, -\infty)$ (because f must be non-increasing), *i.e.* it can be described by the sequence

$$(\delta_0, f(0)) \xrightarrow{\prec_1 c_1 - \delta p_1} (\delta_1, f(\delta_1)) \xrightarrow{\prec_2 c_2 - \delta p_2} \dots (\delta_k, f(\delta_k)) \xrightarrow{<-\infty} \infty\,.$$

In this case, we say that f contains an initial split point at $\delta = 0$, $k - 1$ finite-finite split points δ_i for $i \in [1, k - 1]$, and one finite-infinite split point δ_k. Indeed, $(<, +\infty)$ is the neutral element of min, the absorbing element of $+$, and not in the image of the closure operation, so a $(<, +\infty)$ affine piece cannot be split by those operations. A piece mapping to $(<, -\infty)$, in contrast, can appear as the last piece of f by applying the closure operation.

A bound $f \in$ PWBounds is *continuous over constants* at all finite-finite split points, meaning that for every split point $\delta_i > 0$ with $f(\delta_i) = (\prec, c)$ between a piece $\prec_i c_i - \delta p_i$ and a piece $\prec_{i+1} c_{i+1} - \delta p_{i+1}$, it holds that $c = c_i - \delta_i p_i = c_{i+1} - \delta_i p_{i+1}$. Moreover, f is also continuous at the initial split point $\delta_0 = 0$ with $f(0) = (\prec, c)$ before a piece $\prec_1 c_1 - \delta p_1$, such that $c = c_1$. Finally, if f contains a piece $(<, -\infty)$, f is continuous at the finite-infinite split point δ_k: $f(\delta_k)$ is either equal to $(\prec, c)$ with $c = c_k + \delta_k p_k$ or to $(<, -\infty)$. In our inductive construction from affine bounds, finite-finite split points appear with the min operation as the intersection of finite pieces, and finite-infinite split

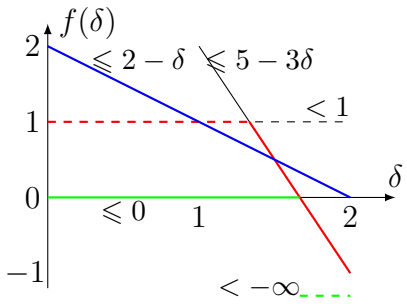


Figure 6.1.: Piecewise affine bounds. Dashed lines represent strict bounds, and infinite bounds are represented at the top and bottom.

points appear with the closure operation.

Example 6.1. Figure 6.1 represents three piecewise affine bounds. The blue bound is made of one piece of equation $\leqslant 2-\delta$. The red bound contains a piece of equation < 1 for $\delta \in [0, 4/3]$ and a piece of equation $\leqslant 5-3\delta$ for $\delta \geqslant 4/3$. The green bound contains a piece of equation $\leqslant 0$ for $\delta \in [0, 5/3]$ and a piece of equation $< -\infty$ for $\delta > 5/3$. If we name the red bound f, then the green bound represents its closure $f^{(*)}$.

Piecewise affine bounds inherit the operations and properties of parametric bounds, most notably they form with min and $+$ an ordered semiring with closure, that we call the *tropical semiring of piecewise affine bounds*.

Lemma 6.2. $(\mathsf{PWBounds}, \min, +)$ *is an ordered semiring with closure.*

Proof. Piecewise affine bounds form an ordered semiring since they are a subset of parametric bounds stable by min and $+$ containing the neutral elements of min and $+$. For every bound $f \in \mathsf{PWBounds}$, it is a parametric bound and we have seen that its closure $f^{(*)}$ in $(\mathsf{Bounds}(\mathbb{R})^{\mathbb{R}_{\geqslant 0}}, \min, +)$ exists. As $\mathsf{PWBounds}$ is stable by closure, $f^{(*)} \in \mathsf{PWBounds}$. Finally, closure in the semiring $(\mathsf{PWBounds}, \min, +)$ is equal to closure in the semiring of parametric bounds, as min (and therefore the order $\sqsubseteq$) and $+$ are identical. □

Every elementary operation (min, $+$, and closure) can be performed in linear time on piecewise affine bounds, and they increase the size of the input in a controlled manner. Formally, if $K, C, P \in \mathbb{N}$, we say that a piecewise affine bound f is bounded by $K[C - \delta P]$ if f contains at most K finite-finite split points and if $|c| \leqslant C$ and $p \leqslant P$ hold for all finite affine pieces of equation $\prec c - \delta p$ in f. If f is bounded by $K[C - \delta P]$, the split points δ_i are non-negative rational numbers whose corresponding irreducible fraction a/b satisfy $a \leqslant 2C$ and $b \leqslant P$ by continuity over constants: finite-finite split points describe the intersection of $c_i - \delta p_i$ and $c_{i+1} - \delta p_{i+1}$ at $\delta = (c_i - c_{i+1})/(p_i - p_{i+1})$, and the finite-infinite split points are created similarly by intersection of an affine piece $c_i - \delta p_i$ with 0. Then, $f(\delta_i) = c_i - \delta_i p_i$ is a rational numbers whose corresponding irreducible fraction a'/b' satisfies $|a'| \leqslant 3CP$ and $b' \leqslant P$. Thus, f can be encoded in size $\mathcal{O}(K \log(PC))$ as

a sequence of split points and affine pieces of the form $(\delta_0, f(0)) \xrightarrow{\prec_1 c_1 - \delta p_1} \dots \infty$ with constants c_i and p_i encoded in binary, and rational pairs $(\delta_i, f(\delta_i))$ encoded as irreducible fractions.

Lemma 6.3. *Let f, g be piecewise affine bounds, such that f is bounded by $K[C - \delta P]$ and g is bounded by $K'[C' - \delta P']$. Then, $\min(f, g)$ is bounded by $(2(K + K') + 1)[\max(C, C') - \delta \max(P, P')]$, $f + g$ is bounded by $(K + K')[(C + C') - \delta(P + P')]$, and $f^{(*)}$ is bounded by $0[0 - \delta 0]$.*

Proof. To bound the constants c and the slopes p in the output of these operations, observe that the equations in the finite pieces of the output are obtained by addition in $f + g$, they cannot change in min, and the closure operation maps every δ to $f(\delta)^{(*)} \in \{(\leqslant, 0), (<, 0), (<, -\infty)\}$. To bound the number of split points after min, observe that new finite-finite split points can only be created by intersection of a finite piece in f and a finite piece in g, such that if we consider the union of the finite-finite split points of f and g, at most one new finite-finite split point appears between two consecutive such split points, plus one after the last split point, giving $K + K' + 1$ new finite-finite split points at most. To bound the number of split points after $+$, observe that the split points in the output of $+$ are at most the union of the split points in its inputs. Finally, the output of the closure operation cannot have finite-finite split points. □

This ensures that every algorithm using these elementary operations polynomially many times will have an exponential time complexity at worst. The driving factor behind this exponential is the number of pieces, that can double when the min or $+$ operations are used.

6.1.2. Piecewise affine DBMs

We construct DBMs over piecewise affine bounds, called piecewise affine DBMs, defined as mappings from $\mathcal{X}_0 \times \mathcal{X}_0$ to $\mathsf{PWBounds}$. A piecewise affine DBM M can equivalently be seen as a mapping from $\delta \in \mathbb{R}_{\geqslant 0}$ to a DBM over $\mathbb{R}$, denoted abusively $\mathsf{M}(\delta)$, such that $(\mathsf{M}(\delta))(\mathsf{x}, \mathsf{y}) = (\mathsf{M}(\mathsf{x}, \mathsf{y}))(\delta)$ for all clocks $\mathsf{x}, \mathsf{y} \in \mathcal{X}_0$.

Piecewise affine DBMs have a normal form that can be computed with a cubic number of elementary operations according to Lemma 1.2.

We show that piecewise affine DBMs are closed under standard operations on zones, and thus the CPre operator can be computed with piecewise affine DBMs:

Lemma 6.4. *Let $e = (\ell, g, \mathcal{Y}, \ell')$ be an edge and M be a piecewise affine DBM. Then, there exists a piecewise affine DBM N, that we can compute using a polynomial number of elementary operations, such that for all $\delta \geqslant 0$, $N(\delta) = \mathsf{CPre}_e^{\delta}(M(\delta))$.*

Proof. We have already defined the addition and minimum of piecewise affine bounds, and if we interpret the operations "set a bound to $(\prec, c)$" as "set it to $\delta \mapsto (\prec, c)$" we can use the standard definitions of $\mathsf{Unreset}$ and intersection from DBMs over $\mathbb{Z}$ [BY04]

A $\mathsf{PreTime}$ operation is defined as replacing for every $\mathsf{x} \in \mathcal{X}$ the entry of index $(\mathsf{x}_0, \mathsf{x})$ in the DBM by $\delta \mapsto (\leqslant, 0)$; replacing for every $\mathsf{x} \in \mathcal{X}$ the entry f of index

$(\mathsf{x}, \mathsf{x}_0)$ by $f + (\delta \mapsto (<, -\delta))$; and normalizing the DBM. A shrinking operation shrink is defined as replacing for every $\mathsf{x} \in \mathcal{X}$ the entries f of index $(\mathsf{x}, \mathsf{x}_0)$ or $(\mathsf{x}_0, \mathsf{x})$ in the DBM by $f + (\delta \mapsto (<, -\delta))$; and normalizing the DBM. We can now define CPre_e as $\mathsf{PreTime}(\mathsf{shrink}[\![g]\!] \cap \mathsf{Unreset}_{\mathcal{Y}}(Z))$. It follows that $\mathsf{CPre}_e(M)(\delta) = \mathsf{CPre}_e^{\delta}(M(\delta))$, because for every $\delta \leqslant 0$, $\mathsf{PreTime}(M)(\delta) = \mathsf{PreTime}_{>\delta}(M(\delta))$, $\mathsf{shrink}(M)(\delta) = \mathsf{shrink}_{[-\delta,\delta]}(M(\delta))$ and $\mathsf{Unreset}_{\mathcal{Y}}(M)(\delta) = \mathsf{Unreset}_{\mathcal{Y}}(M(\delta))$. The elementary operations on piecewise affine bounds are only called polynomially many times at most while computing CPre_e. □

The main difference with classical DBMs lies in the emptiness check: while a classical (or shrunk) DBM can either be empty or not, this notion does not exist for piecewise affine DBMs. Indeed, after normalization, a piecewise affine DBM M may contain diagonal entries not comparable with zero (zero is the bound $\delta \mapsto (\leqslant, 0)$). For example, the green bound from Figure 6.1 is a bound incomparable with zero that can be output by the closure operation. One can compute for every bound f on a diagonal entry of the DBM some $\delta_f = \sup\{\delta \mid (\leqslant, 0) \preccurlyeq f(\delta)\}$, and then consider δ_d, the min of those δ_f. Then, it follows that δ_d is the supremum of δ such that $M(\delta)$ is not empty. In contrast, shrunk DBMs perform the same computations as piecewise affine DBMs, but as they are solely interested in expressing statements about $\delta > 0$ small enough, they only keep in memory the first affine piece of our piecewise affine bounds.

We finally define an operator CPre_π on piecewise affine DBMs M, such that for all $\delta \geqslant 0$, $\mathsf{CPre}_\pi(M)(\delta) = \mathsf{CPre}_e^{\delta}(M(\delta))$. This is done by induction on the length of π: if $\pi = e\pi'$ with e and edge of $\mathcal{A}$, then

$$\mathsf{CPre}_\pi^{\delta}(M(\delta)) = \mathsf{CPre}_e^{\delta}(\mathsf{CPre}_{\pi'}^{\delta}(M(\delta)))\,,$$

and Lemma 6.4 allows us to conclude; and if π is of length 0,

$$\mathsf{CPre}_\pi(M)(\delta) = M(\delta)\,.$$

6.2. Largest admissible perturbation of a lasso

We have seen that a symbolic computation of the robust predecessors of a finite path, that holds for all values of δ, can be computed with piecewise affine DBMs. Consider now a cycle π in a timed automaton $\mathcal{A}$. We will now explain how to compute the greatest fixpoint of CPre over π with piecewise affine DBMs.

Lemma 6.5. *Let π be a cycle. There exists a piecewise affine DBM N, that we can compute using a polynomial number of elementary operations, such that for all $\delta \geqslant 0$, $N(\delta) = \nu X\, \mathsf{CPre}_\pi^{\delta}(X)$.*

Proof. Let n equal $2|\mathcal{X}_0|^2$. We can compute piecewise affine DBMs M_1 and M_2 encoding respectively $\mathsf{CPre}_{\pi^n}(\top)$ and $\mathsf{CPre}_{\pi^{n+1}}(\top)$. Then, by monotonicity, the inclusion $M_1(\delta) \subseteq M_2(\delta)$ holds for every $\delta \geqslant 0$, and if $M_1(\delta) = M_2(\delta)$, then $\nu X\, \mathsf{CPre}_\pi^{\delta}(X) = M_1(\delta)$. By Proposition 5.3, for every $\delta \geqslant 0$, if $M_1(\delta) \subsetneq M_2(\delta)$, then $\nu X\, \mathsf{CPre}_\pi^{\delta}(X) = \emptyset$. In addition, both M_1 and M_2 are non-increasing w.r.t. δ, thus one can identify in linear time the

value $\delta_i = \inf\{\delta \geq 0 \mid M_1(\delta) \subsetneq M_2(\delta)\}$. Then, let N be $\min(M_1, M_{\delta_i})$, with M_{δ_i} the piecewise affine DBM where every entry $M_{\delta_i}(\mathsf{x}, \mathsf{y})$ maps δ to $M_1(\mathsf{x}, \mathsf{y})(\delta)$ if $\delta < \delta_i$ and to $(<, -\infty)$ if $\delta > \delta_i$. The value of bounds in $M_{\delta_i}(\mathsf{x}, \mathsf{y})$ for $\delta = \delta_i$ is similarly set to $M_1(\mathsf{x}, \mathsf{y})$ if $M_1(\delta_i) = M_2(\delta_i)$ and to $(<, -\infty)$ otherwise. □

As a consequence, we obtain the following new result:

Proposition 6.1. *We can compute the largest admissible perturbation of a lasso in exponential time.*

Proof. Let $\pi_1\pi_2^\omega$ be a lasso. As a first step, we compute the greatest fixpoint of CPre_{π_2} as a piecewise affine DBM. As a second step, one applies the operator CPre_{π_1} to the greatest fixpoint. We denote the result by M. To conclude, one can then return the value $\sup\{\delta \mid \mathbf{0} \in M(\delta)\}$ of maximal perturbation. The elementary operations min and $+$ are only used polynomially many times in this computation and thus all piecewise affine bounds used as inputs for elementary operations are bounded by $K[C - \delta P]$, with K, C and P at most exponential in the size of the automaton. □

Part III.

Weighted timed games

Introduction

In this part we study problems related to the *optimality* of controllers. Let us start by studying finite weighted games equipped with a reachability objective.

Finite weighted games

The context is that of a two-player turn-based game where transitions have been equipped with weights. The optimal reachability problem asks what is the lowest cumulated weight that controller can guarantee for reaching a target from a given intial state, against any decisions made by the antagonistic environment. The controller is therefore the minimiser player, while the environment wants to maximise the weight accumulated along the execution. This lowest weight is called the *value* of the game, and computing it can be seen as a natural generalisation of the classical shortest path problem in a weighted graph to the case of two-player games. Weighted games with reachability objectives have been recently explored as a refinement of mean-payoff games [BGHM15, BGHM16].

We focus particularly on challenges related to having *negative weights*, that are crucial when one wants to model energy or other resources that can grow or decrease during an execution of the system to study. If weights of transitions are all non-negative, a generalised Dijkstra algorithm allows one to compute the value in polynomial time [KBB+08]. In the presence of negative weights, a pseudo-polynomial time solution (*i.e.* polynomial if weights are encoded in unary, but exponential otherwise) has been given in [BGHM16]. Moreover, deciding if the value is equal to $-\infty$ is shown to be in $\mathsf{NP} \cap \mathsf{coNP}$, and as hard as solving mean-payoff games. Having negative weights therefore comes with a price on the complexity side.

In Chapter 7, we introduce the *value iteration* algorithm used by [BGHM16], then study classes of weighted games where the value can be computed in polynomial time, even in the presence of negative weights encoded in binary. These ideas will be lifted to the timed setting in later chapters, and will provide notable classes of weighted timed games.

Our contribution is to define *divergent* and *almost-divergent* weighted games, and show that they are solvable in polynomial time. To our knowledge, they are the first non-trivial classes of weighted games with negative weights solvable in polynomial time, apart from the class of acyclic games, that they contain. The intuition behind these classes can be expressed as follows:

- In divergent games, cycles of total cumulated weight 0 are forbidden. As a result, the weight of long execution must diverge to either $+\infty$ or $-\infty$, and we can use this information to speed up the value iteration algorithm of [BGHM16].

Untimed	weights in $\mathbb{N}$	weights in $\mathbb{Z}$		
	all games	divergent	almost-div.	all games
Value pb.	PTIME [KBB+08]	PTIME-complete Thm. 7.1,7.2		pseudo-poly. [BGHM16]
Value $-\infty$	/	PTIME-complete Prop. 7.4, Lm. 7.4		pseudo-poly. [BGHM16]
Value $+\infty$	PTIME [KBB+08]	PTIME-complete Prop. 7.1, [BGHM16]		
Membership	/	NL-complete (unary wt.) PTIME (binary wt.) Thm. 7.1,7.2		/

Table 1.: Solving weighted games with arbitrary weights

- This logic is then extended to almost-divergent games, where only some "bad" cycles of weight 0 are forbidden, based on a stability by decomposition rule (it is forbidden for a cycle of weight 0 to be the composition of a positive cycle and a negative one). In this case, inspired by [BJM15], we define a notion of kernel of the game, that contains all cycles of weight 0, and deal with them separately.

We also solve the *membership* problem, *i.e.* check if an instance belongs to these classes, in polynomial time. Table 1 summarises our results. The value problem compares the value to a given threshold α, and particular cases of interest are the thresholds $-\infty$ and $+\infty$.

Weighted timed games

We then turn our attention to weighted games played on a timed automaton instead of a finite transition system. The resulting optimal reachability problems take place on a *weighted timed automaton* [BFH+01, ALTP04], with weights on edges, and rates on locations that let weight accumulate linearly when time elapses. If one does not care about optimality (and weights), the problem reduces to solving reachability timed games. This problem is EXPTIME-complete [JT07], and can be solved efficiently with symbolic techniques (see Uppaal Tiga [CDF+05]). However, the quantitative setting is notoriously difficult: While solving weighted timed automata has been shown to be PSPACE-complete [BBBR07], weighted timed games are known to be undecidable [BBR05].

Faced with this challenge, several approaches have been followed. A semi-algorithm, extending the *value iteration* algorithm of weighted games to the timed setting, has been described in [BCFL04]. It takes the form of a greatest fixpoint computation, that may not terminate. Alternatively, once can study restricted classes of weighted timed games in order to regain decidability.

The most restrictive one is the class of *acyclic* games, where the dynamic has bounded

length, *i.e.* controller must reach a target within a fixed number of steps, or be punished with a weight of $+\infty$. The resulting problem has been shown to be decidable, and can be solved in double-exponential time [TMM02]. In fact, the semi-algorithm of [BCFL04] also solves acyclic weighted timed games, as their greatest fixpoint computation must converge within a fixed number of steps. This result has been further refined in [ABM04], where a fairly involved analysis claimed that the value iteration algorithm could be implemented in exponential time.

It is worth noting that all of these results are stated for weighted timed games with only non-negative weights (in edges and locations). Less is known for weighted timed games in the presence of negative weights, and to our knowledge no results exist so far for a class where the underlying timed automata have more than one clock. While the value iteration algorithm can allow for the presence of negative weights (it is still a semi-algorithm, and the fixpoint arguments did not require non-negative weights), we argue that the complexity analysis of [ABM04] should be put into question. To this end, we will detail a value iteration procedure for solving acyclic weighted timed games with arbitrary weights. It follows closely the techniques of [ABM04], but is more symbolic.[1] However, we could not replicate their complexity analysis, and will therefore rely on a double-exponential time upper bound instead of an exponential one (see Chapter 10 for detailed explanations). This will induce an exponential gap between the results of non-negative weights and our corresponding results for arbitrary weights.

In the context of non-negative weights, the largest class that enjoys decidability results is defined by the strictly non-Zeno cost hypothesis [BCFL04]: it states that every execution of the timed automaton that follows a cycle of the region abstraction has a weight far from 0 (in interval $[1, +\infty)$, for instance). Indeed, it has been shown in [BCFL04] that under this hypothesis controller can be restricted to reaching a target fixed number of steps, exponential in the size of the game. In combination with [TMM02], this leads to a triply-exponential time decision procedure, lowered to 2-EXPTIME with [ABM04].

We introduce a generalisation of the strictly non-Zeno cost hypothesis in the presence of negative weights, that we call *divergence*. In the class of divergent weighted timed games, each execution that follows a cycle of the region abstraction must have weight far from 0, *i.e.* in $(-\infty, -1] \cup [1, +\infty)$. We show the decidability of this class, with a 3-EXPTIME complexity (and an EXPTIME-hardness lower bound). The decision procedure relies on showing that controller can be restricted to almost-optimal strategies that reach a target in a bounded number of steps, exponential in the size of the game, matching asymptotically the horizon that could be obtained in the non-negative case from [BCFL04].

As in the untimed setting, we relax our requirement slightly and define a larger class, of *almost-divergent* weighted timed games. This time, executions that follow a cycle of the region abstraction must have weight in $(-\infty, -1] \cup \{0\} \cup [1, +\infty)$, and those of weight 0 are subject to a stability by decomposition requirement. On this class, the decision procedure of divergent games cannot be extended, as the value problem becomes undecidable [BJM15]. This negative result already holds in the presence of non-negative

[1] It performs computations on the entire state-space at once, whereas [ABM04] required constructing the entire region abstraction.

weights only. For this case (where cycles have weight in $\{0\} \cup [1, +\infty)$), it has been proposed an approximation schema to compute arbitrarily close estimates of the optimal value [BJM15]. To this end, the authors consider regions with a refined granularity so as to control the precision of the approximation.

Our main result on almost-divergent weighted timed games can be stated as follows: Given an almost-divergent weighted timed game, an initial configuration c and a threshold ε, we can compute a value that we guarantee to be ε-close to the optimal value when the play starts from c.

In order to approximate almost-divergent weighted timed games, we first adapt the approximation schema of [BJM15] to our setting. At the very core of their schema is the notion of *kernels* that collect all cycles of weight exactly 0 in the game. Then, a semi-unfolding of the game (in which kernels are not unfolded) of bounded depth is shown to be equivalent to the original game. Adapting this schema to negative weights requires to address new issues:

- The definition and the approximation of these kernels is much more intricate in our setting. Indeed, with only non-negative weights, a cycle of weight 0 only encounters locations and edges with weight 0. It is no longer the case with arbitrary weights, both for discrete weights on edges (that could alternate between weight $+1$ and -1, *e.g.*) and continuous rates on locations: for this continuous part, this requires to keep track of the real-time dynamics of the game.

- Some configurations may have value $-\infty$. While it is undecidable in general whether a configuration has value $-\infty$, we prove that it is decidable for almost-divergent weighted timed games.

- The identification of an adequate bound to define an equivalent semi-unfolding of bounded depth is more difficult in our setting, as having guarantees on weight accumulation is harder (we can lose accumulated weight). We deal with this by evaluating how large the value of a configuration can be, provided it is not infinite.

We also develop a more symbolic approximation schema, in the sense that it avoids the a priori refinement of regions. Instead, all computations are performed in a symbolic way using a single call to the value iteration algorithm. This allows to mutualise as much as possible the different computations: comparing these schemas with the evaluation of MDPs or quantitative games like mean-payoff or discounted-payoff, it is the same improvement as when using value iteration techniques instead of techniques based on the unfolding of the model into a finite tree which can contain many times the same location.

This result can also be interpreted as a bound on the convergence speed of the fixpoint computation performed by value iteration. Moreover, we prove that deciding if a weighted timed game is divergent / almost-divergent is a PSPACE-complete problem.

Table 2 summarises our results on weighted timed games.

Other types of payoffs than the accumulated weight we study (*i.e.* total payoff) have been considered for weighted timed games. For instance, energy and mean-payoff timed games have been introduced in [BCR14]. They are also undecidable in general.

Timed	weights in $\mathbb{N}$		
	divergent	almost-divergent	all WTG
Value pb.	2-EXPTIME [BCFL04]+[ABM04]	undecidable [BJM15]	undecidable [BBR05]
Approx. pb.	/	2-EXPTIME [BJM15]+[ABM04]	?
Value $+\infty$	EXPTIME-complete [BCFL04]		

Timed	weights in $\mathbb{Z}$		
	divergent	almost-divergent	all WTG
Value pb.	3-EXPTIME EXPTIME-hard Thm. 9.1	undecidable [BJM15]	undecidable [BGNK$^+$14]
Approx. pb.	/	3-EXPTIME Thm. 9.2	?
Value $-\infty$	EXPTIME-complete Prop. 10.2,9.1		undecidable Prop. 9.2
Value $+\infty$	EXPTIME-complete Prop. 7.1		
Membership	PSPACE-complete Thm. 9.1,9.2		/

Table 2.: Solving weighted timed games with arbitrary weights

Interestingly, a subclass called *robust timed games*, not far from our divergence hypothesis, admits decidability results for other payoffs. A weighted timed game is robust if, to say short, every simple cycle (cycle without repetition of a state) has weight non-negative or less than a constant $-\varepsilon$. Solving robust timed game can be done in EXPSPACE, and is EXPTIME-hard. Moreover, deciding if a weighted timed game is robust has complexity 2-EXPSPACE (and coNEXPTIME-hard). This contrasts with our PSPACE results for the membership problem.[2] It has to be noted that extending our techniques and results in the case of robust timed games is intrinsically not possible: indeed, the value problem for this class is undecidable [BJM15].

This part is structured as follows: We start by presenting our results on finite optimal reachability games in Chapter 7. We introduce weighted timed games in Chapter 8, and define the divergent and almost-divergent classes in Chapter 9. Moreover, in Chapter 9 we study structural properties of these classes, and state our main results. Our decidability results are then proven in Chapter 10, and finally approximation techniques are presented in Chapter 11.

[2]While our divergent games have a similar definition, both classes are incomparable.

7. Finite weighted games

7.1. The untimed setting

We consider two-player turn-based games played on weighted transition systems and denote the two *players* by Min and Max instead of Ctrl and Env.

Definition 7.1. A *weighted game*[1] is a tuple $\mathcal{G} = \langle S_{\mathsf{Min}}, S_{\mathsf{Max}}, S_{\mathsf{t}}, \Sigma, T, \mathsf{wt}, \mathsf{wt}_{\mathsf{t}} \rangle$ where $\langle S_{\mathsf{Min}}, S_{\mathsf{Max}}, T \rangle$ is a two-player turn-based game labelled over Σ, of states $S = S_{\mathsf{Min}} \uplus S_{\mathsf{Max}}$ and transitions $T \subseteq S \times \Sigma \times S$, $\mathsf{wt} \colon T \to \mathbb{Z}$ is the weight function associating an integer weight with each transition[2], $S_{\mathsf{t}} \subseteq S_{\mathsf{Min}}$ is a subset of target states for player Min, and $\mathsf{wt}_{\mathsf{t}} \colon S_{\mathsf{t}} \to \mathbb{Z}_\infty$ is a function mapping each target state to a final weight of $\mathbb{Z}_\infty = \mathbb{Z} \cup \{-\infty, +\infty\}$.

The addition of final weights is not standard, but we will use it in the process of solving those games: In any case, it is possible to simply map each target state to the weight 0, allowing us to recover the standard definition. These games need not be finite in general, but in Chapter 7, we limit our study to the resolution of finite weighted games (where all previous sets are finite). We suppose that: (i) the game is deadlock-free except on target states,[3] *i.e.* for each state $s \in S \backslash S_{\mathsf{t}}$, there is a letter $a \in \Sigma$ and a state $s' \in S$, such that $(s, a, s') \in T$, and there are no transitions starting from states in S_{t}; (ii) the game is deterministic, *i.e.* for each pair $(s, a) \in S \times \Sigma$, there is at most one state $s' \in S$ such that $(s, a, s') \in T$.

Notions of maximal (resp. non-maximal) plays, strategies and their outcome are derived from the two-player turn-based game $\langle S_{\mathsf{Min}}, S_{\mathsf{Max}}, T \rangle$ as in Chapter 1.3:

- A play is a sequence of consecutive transitions, it is non-maximal if it is finite and ends in $S \backslash S_{\mathsf{t}}$ and maximal if it is infinite or ends in S_{t};
- A strategy for player Min (resp. player Max) is a mapping $\sigma_{\mathsf{Min}} \colon \mathsf{FPlays}^{\mathsf{Min}} \to \Sigma$ (resp. $\sigma_{\mathsf{Max}} \colon \mathsf{FPlays}^{\mathsf{Max}} \to \Sigma$) from non-maximal plays ending in a state of Min (resp. Max) to labels;
- $\mathsf{play}(s, \sigma_{\mathsf{Min}}, \sigma_{\mathsf{Max}})$ denotes the unique maximal play starting in s and conforming to σ_{Max} and σ_{Min}.

[1] Weighted games are called *min-cost reachability games* in [BGHM16].

[2] If $t = s \xrightarrow{a} s'$ is a transition in $\mathcal{G}$, we denote $\mathsf{wt}(s, a, s')$ the weight $\mathsf{wt}(t)$.

[3] this comes without loss of generality, as any non-target deadlock state can become a target state of final weight $+\infty$.

Recall that the finite play ρ is said to be a *cycle* if it ends in its first state—*i.e.* $\mathsf{first}(\rho) = \mathsf{last}(\rho)$—and if $|\rho| > 0$. Recall also that the strategy σ is said *positional* if for all non-maximal plays ρ, ρ' ending in the same state, we have that $\sigma(\rho) = \sigma(\rho')$.

The objective of Min is to reach a target state, while minimising the accumulated weight up to the target. Hence, we associate to every finite play $\rho = s_0 \xrightarrow{a_1} s_1 \dots \xrightarrow{a_k} s_k$ its cumulated weight

$$\mathsf{wt}_\Sigma(\rho) = \sum_{i=0}^{k-1} \mathsf{wt}(s_i, a_{i+1}, s_{i+1}) \,.$$

Then, the weight of a maximal play ρ, also denoted by $\mathsf{wt}(\rho)$, is defined by $+\infty$ if ρ is infinite and thus does not reach S_t, and $\mathsf{wt}_\Sigma(\rho) + \mathsf{wt}_\mathsf{t}(s_\mathsf{t})$ if it is finite and ends in $s_\mathsf{t} \in S_\mathsf{t}$. Then, we let $\mathsf{Val}(s, \sigma_\mathsf{Min})$ and $\mathsf{Val}(s, \sigma_\mathsf{Max})$ be the respective values of the strategies:

$$\begin{aligned} \mathsf{Val}(s, \sigma_\mathsf{Min}) &= \sup_{\sigma_\mathsf{Max}} \mathsf{wt}(\mathsf{play}(s, \sigma_\mathsf{Min}, \sigma_\mathsf{Max})) \\ \mathsf{Val}(s, \sigma_\mathsf{Max}) &= \inf_{\sigma_\mathsf{Min}} \mathsf{wt}(\mathsf{play}(s, \sigma_\mathsf{Min}, \sigma_\mathsf{Max})) \,. \end{aligned}$$

Finally, for all states s, we let $\underline{\mathsf{Val}}(s) = \sup_{\sigma_\mathsf{Max}} \mathsf{Val}(s, \sigma_\mathsf{Max})$ and $\overline{\mathsf{Val}}(s) = \inf_{\sigma_\mathsf{Min}} \mathsf{Val}(s, \sigma_\mathsf{Min})$ be the *lower* and *upper values* of s, respectively. We may easily show that $\underline{\mathsf{Val}}(s) \leqslant \overline{\mathsf{Val}}(s)$ for all s. We say that strategies $\sigma^\star_\mathsf{Min}$ of Min and $\sigma^\star_\mathsf{Max}$ of Max are optimal if, for all states s, $\mathsf{Val}(s, \sigma^\star_\mathsf{Max}) = \underline{\mathsf{Val}}(s)$ and $\mathsf{Val}(s, \sigma^\star_\mathsf{Min}) = \overline{\mathsf{Val}}(s)$, respectively. We say that a game $\mathcal{G}$ is *determined* if for all states s, its lower and upper values are equal. In that case, we write $\mathsf{Val}(s) = \underline{\mathsf{Val}}(s) = \overline{\mathsf{Val}}(s)$, and refer to it as the *value* of s in $\mathcal{G}$. Finite weighted games are known to be determined [BGHM16]. If the game is not clear from the context, we may add an index $\mathcal{G}$ to previous notations, such that $\mathsf{wt}_\mathcal{G}(\rho)$ is the weight of a maximal play ρ in $\mathcal{G}$ (taking into account both cumulated and final weights), and $\mathsf{Val}_\mathcal{G}(s)$ is the value of s in $\mathcal{G}$.

We denote by $w_{\max}$ the maximal weight in absolute value of transitions in $\mathcal{G}$, such that $w_{\max} = \max_{t \in T} |\mathsf{wt}(t)|$. Similarly, we let $w^\mathsf{t}_{\max}$ denote the maximal finite final weight of target states in $\mathcal{G}$:

$$w^\mathsf{t}_{\max} = \max\{|\mathsf{wt}_\mathsf{t}(s_\mathsf{t})| \mid s_\mathsf{t} \in S_\mathsf{t} \wedge \mathsf{wt}_\mathsf{t}(s_\mathsf{t}) \in \mathbb{Z}\} \,.$$

An algorithm $\mathcal{A}$ of input $\mathcal{G}$ whose time-complexity is polynomial in $|S|$, $w_{\max}$ and $w^\mathsf{t}_{\max}$ is called pseudo-polynomial. Indeed, its complexity relative to the size of the input depends on the encoding of weights: $\mathcal{A}$ is a polynomial-time algorithm if weights and final weights are encoded in unary, but is exponential if they are encoded in binary (as $w_{\max}$ and $w^\mathsf{t}_{\max}$ could be exponential in the size of $\mathcal{G}$). In contrast, $\mathcal{A}$ is said polynomial if it can be run in polynomial time when weights are encoded in binary.

7.1.1. Problems

We want to compute the value of a *finite* weighted game, as well as optimal strategies for both players, if they exist. The corresponding decision problem, called the *value*

problem, asks whether $\mathsf{Val}_{\mathcal{G}}(s) \leqslant \alpha$, given a finite weighted game $\mathcal{G}$, one of its states s, and a threshold $\alpha \in \mathbb{Z}$. Other thresholds can also be considered, *e.g.* $\mathsf{Val}_{\mathcal{G}}(s) \bowtie \alpha$ with $\bowtie \in \{<, >, \leqslant, \geqslant, =, \neq\}$.[4] We also define the value problem with threshold $+\infty$ (resp. $-\infty$), that asks, given a finite weighted game $\mathcal{G}$ and one of its states s, whether $\mathsf{Val}_{\mathcal{G}}(s) = +\infty$ (resp. $-\infty$). Every upper bound that we will mention (about the time-complexity required to solve the value problem) relies on computing the value of every state, and therefore on solving all of these problems equivalently. The lower bounds results we will mention refer to the $\leqslant \alpha$ threshold.

7.2. Solving weighted games

Let $\mathcal{G}$ be a finite weighted game. We say that $\mathcal{G}$ is acyclic if it does not contain cycles, and use standard notions for directed acyclic graphs: there exists a topological order on states where s is smaller than s' if and only if there exists a path from s to s'. We call roots the smallest states, and leaves the greatest. For every state s, the sub-game of root s refers to the game whose states are all states greater than or equal to s, and whose transitions and weights match those of $\mathcal{G}$. Every path in an acyclic game is finite of length bounded by $|S| - 1$, and we let the depth of $s \in S$ (resp. the depth of $\mathcal{G}$) be the greatest length of paths from a root to s (resp. from a root to a leaf).

If $\mathcal{G}$ is acyclic then, by the no-deadlocks assumption, the target states are exactly the leaves of $\mathcal{G}$. The value of a state s in $\mathcal{G}$ depends solely on weights and transitions in the sub-game of root s. Indeed, if the value of every state in this sub-game, excluding s, is already known, the value of s can be computed, as

$$\mathsf{Val}(s) = \begin{cases} \min_{(s,a,s') \in T} \left[\mathsf{wt}(s,a,s') + \mathsf{Val}(s')\right] & \text{if } s \in S_{\mathsf{Min}} \\ \max_{(s,a,s') \in T} \left[\mathsf{wt}(s,a,s') + \mathsf{Val}(s')\right] & \text{if } s \in S_{\mathsf{Max}}\,. \end{cases}$$

The value of every state in $\mathcal{G}$ can thus be computed in a bottom-up fashion (*i.e.* in reverse topological order), as the value of leaves equals the final weight wt_{t}.

This technique can be extended to weighted games containing cycles in the form of a fixpoint computation, that we will call the value iteration algorithm.

7.2.1. Value iteration

If $\boldsymbol{V}$ represents a value function—*i.e.* a mapping from states of S to a value in $\mathbb{Z} \cup \{+\infty, -\infty\}$—we denote by $\boldsymbol{V}_s$ the image $\boldsymbol{V}(s)$, for better readability. One step of the game is summarised in the following operator $\mathcal{F}$ mapping each value function $\boldsymbol{V}$ to a value function $\boldsymbol{V}' = \mathcal{F}(\boldsymbol{V})$ defined by $\boldsymbol{V}'_s = \mathsf{wt}_{\mathsf{t}}(s)$ if $s \in S_{\mathsf{t}}$, and otherwise

$$\boldsymbol{V}'_s = \begin{cases} \min_{s \xrightarrow{a} s'} \left[\mathsf{wt}(s,a,s') + V_{s'}\right] & \text{if } s \in S_{\mathsf{Min}} \\ \max_{s \xrightarrow{a} s'} \left[\mathsf{wt}(s,a,s') + V_{s'}\right] & \text{if } s \in S_{\mathsf{Max}} \end{cases} \tag{7.1}$$

[4] Since $\mathsf{Val}_{\mathcal{G}}(s) \in \mathbb{Z}_{\infty}$, these other thresholds reduce to multiple instances of the value problem with threshold $\leqslant \alpha$, $\leqslant \alpha - 1$, or its negation with threshold $> \alpha$.

where $s \xrightarrow{a} s'$ ranges over the transitions in $\mathcal{G}$ that start from s. Then, starting from $\boldsymbol{V}^0$ mapping every state to $+\infty$, except for the targets mapped to their final weight according to wt_t, we let

$$\boldsymbol{V}^i = \mathcal{F}(\boldsymbol{V}^{i-1})$$

for all $i > 0$. The value function $\boldsymbol{V}^i$ represents the value Val^i, which is intuitively what Min can guarantee when forced to reach the target in at most i steps.

More formally, we define $\mathsf{wt}^i(\rho)$ the weight of a maximal play ρ at horizon i, as $\mathsf{wt}(\rho)$ if ρ reaches a target state in at most i steps, and $+\infty$ otherwise. Using this alternative definition of the weight of a play, we can obtain a new game value

$$\mathsf{Val}^i(s) = \inf_{\sigma_\mathsf{Min}} \sup_{\sigma_\mathsf{Max}} \mathsf{wt}^i(\mathsf{play}(s, \sigma_\mathsf{Min}, \sigma_\mathsf{Max}))\,.$$

Notice that $\mathcal{F}$ is a monotonic operator, *i.e.* if $\boldsymbol{V}, \boldsymbol{V}'$ are two value functions such that $\boldsymbol{V}_s \geqslant \boldsymbol{V}'_s$ for all states $s \in S$, then $\forall s \in S, \mathcal{F}(\boldsymbol{V})_s \geqslant \mathcal{F}(\boldsymbol{V}')_s$. Moreover $\forall s \in S, \boldsymbol{V}^0_s \geqslant \mathcal{F}(\boldsymbol{V}^0)_s$ since $\boldsymbol{V}^0$ maps every non-target state to $+\infty$, and target state keep the same value. It follows that the sequence $(\boldsymbol{V}^i)_{i\in\mathbb{N}}$ is non-increasing on every entry, as $\boldsymbol{V}^i = \mathcal{F}^i(\boldsymbol{V}^0) \geqslant \mathcal{F}^i(\mathcal{F}(\boldsymbol{V}^0)) = \boldsymbol{V}^{i+1}$. The value iteration algorithm consists in finding the greatest fixpoint of operator $\mathcal{F}$, *i.e.* the limit of the sequence $(\boldsymbol{V}^i)_{i\in\mathbb{N}}$. Indeed, this greatest fixpoint is known to be the vector of values of the game (see, *e.g.*, [BGHM16, Corollary 11]).

Then, if $\mathcal{G}$ is an acyclic game of depth d, the fixpoint is reached after d steps, and $\mathsf{Val} = \boldsymbol{V}^d$. In this case, the infinite values in $\mathcal{G}$ (*i.e.* states s with $\mathsf{Val}(s) \in \{-\infty, +\infty\}$) are derived from reaching targets with infinite final weights.

If the game contains cycles, infinite values can also come from arbitrarily long plays: a state s can have value $+\infty$ if Max can force an infinite play, never reaching any target, and it can have value $-\infty$ if Min can enforce an arbitrarily low weight, *e.g.* by staying in a cycle of negative cumulated weight. These $+\infty$ states correspond to a safety objective for player Max, and can be computed in polynomial time: it is shown in [BGHM16] that for all $s \in S$, $\mathsf{Val}(s) = +\infty$ if and only if $\boldsymbol{V}_s^{|S|} = +\infty$. In contrast, deciding if a state has value $-\infty$ has no known polynomial solution (it is as hard as solving mean-payoff games). In [BGHM16], it is shown that in the presence of negative weights the sequence $(\boldsymbol{V}^i)_{i\in\mathbb{N}}$ stabilises after a number of iterations pseudo-polynomial on states with value in $\mathbb{R} \cup \{+\infty\}$, and that states with value $-\infty$ can be detected in this computation (they are those where the computed value goes under a given threshold).

7.2.2. Optimal strategies

Let us fix an initial state s. By definition of lower and upper values, there exists for each player $\mathsf{P} \in \{\mathsf{Min}, \mathsf{Max}\}$ a sequence of strategies $(\sigma^i_\mathsf{P})_{i\in\mathbb{N}}$ such that $\lim_{i\to\infty} \mathsf{Val}(s, \sigma^i_\mathsf{P}) = \mathsf{Val}(s)$ and such that the sequence $(\mathsf{Val}(s, \sigma^i_\mathsf{Min}))_{i\in\mathbb{N}}$ is non-increasing over $\mathbb{Z}_\infty$, while $(\mathsf{Val}(s, \sigma^i_\mathsf{Max}))_{i\in\mathbb{N}}$ is non-decreasing. If the sequence $(\mathsf{Val}(s, \sigma^i_\mathsf{P}))_{i\in\mathbb{N}}$ stabilizes for all $i \geqslant k$, then σ^k_P is an optimal strategy of player P for s, *i.e.* $\mathsf{Val}(s, \sigma^k_\mathsf{P}) = \mathsf{Val}(s)$. Therefore, if $\mathsf{Val}(s) > -\infty$ then Min must have an optimal strategy for s (an infinite decreasing

sequence over $\mathbb{Z}$ stabilizes), and if $\mathsf{Val}(s) < +\infty$ then Max has an optimal strategy for s. Moreover, if an optimal strategy for P exists for all states s, then they can be combined into an (overall) optimal strategy for P.

In fact, there always exists a *positional* strategy $\sigma^\star_{\mathsf{Max}}$ for Max that is optimal, even if some states have value $+\infty$. The strategy $\sigma^\star_{\mathsf{Max}}$ can be obtained in the value iteration algorithm, by memorizing for every state $s \in S_{\mathsf{Max}}$ the transition that maximizes $\max_{s \xrightarrow{a} s'} \left[\mathsf{wt}(s,a,s') + V_{s'}\right]$ in the last application of $\mathcal{F}$. However, this does not hold for Min, as there might be no sequence of positional strategies for player Min whose value at s converges towards $\mathsf{Val}(s)$. In [BGHM16], it is shown that value iteration can also compute an optimal strategy for Min (or a sequence of strategies in the $-\infty$ case), by switching between two positional strategies $\sigma^\star_{\mathsf{Min}}$ and $\sigma^\dagger_{\mathsf{Min}}$: $\sigma^\star_{\mathsf{Min}}$ accumulates negative weight by following negative cycles, and $\sigma^\dagger_{\mathsf{Min}}$ ensures reaching a target. The optimal strategy of Min follows the decisions of $\sigma^\star_{\mathsf{Min}}$, until switching to the decisions of $\sigma^\dagger_{\mathsf{Min}}$ when the length of the play is greater than a finite bound k. These strategies thus require finite memory in the form of a counter.

An interesting case happens if $\mathcal{G}$ has no cycles of negative cumulated weight, *e.g.* if weights are non-negative.

Lemma 7.1. *If $\mathcal{G}$ has no cycles of negative cumulated weight, then both players have optimal strategies that are positional. Moreover, $\mathsf{Val}_\mathcal{G} = \mathsf{Val}^{|S|}_\mathcal{G}$ and the optimal strategies can be computed in polynomial time.*

Proof. Any strategy of Min is optimal on states of value $+\infty$, so we will ignore those. As there are no negative cycles, value $-\infty$ can only be obtained through reaching a target with final weight $-\infty$. As a consequence, Min has an optimal strategy σ_{Min} that switches between $\sigma^\star_{\mathsf{Min}}$ and $\sigma^\dagger_{\mathsf{Min}}$ when the length of the play is greater than some k, as detailed in [BGHM16]. The strategy $\sigma^\star_{\mathsf{Min}}$ is only compatible with cycles of negative cumulated weight, and $k \geqslant |S| + 1$, therefore $\sigma^\dagger_{\mathsf{Min}}$ is never used. Thus, Min has an optimal strategy $\sigma^\star_{\mathsf{Min}}$ that is positional. Then, consider $\mathsf{Val}^{|S|}_\mathcal{G}$, the value with bounded horizon $|S|$. For every state s, we have $\mathsf{Val}^{|S|}_\mathcal{G}(s, \sigma^\star_{\mathsf{Min}}) = \mathsf{Val}_\mathcal{G}(s)$, so that $\mathsf{Val}^{|S|}_\mathcal{G} \leqslant \mathsf{Val}_\mathcal{G}$. As $\mathsf{Val}_\mathcal{G} \leqslant \mathsf{Val}^k_\mathcal{G}$ holds for any $k \geq 0$, $\mathsf{Val}_\mathcal{G} = \mathsf{Val}^{|S|}_\mathcal{G}$. Finally, since value iteration converges in $|S|$ steps, the computation of optimal strategies described in [BGHM16] runs in polynomial time. □

7.2.3. Safely removing states of infinite value

In this chapter, we study classes of weighted games with arbitrary weights where values can be computed in polynomial time. As a first step, we explain how to compute and remove states with infinite value in polynomial time. The only states with infinite value that will remain are some states that derive a value of $-\infty$ from arbitrary accumulation of negative weight.

Let us start by formalising a way to safely remove states whose value is known to be infinite. Let $S^{-\infty}$ be a subset of S, such that $\mathsf{Val}(s) = -\infty$ for all $s \in S^{-\infty}$. If a state of $\mathcal{G}$ is in the attractor of Min towards $S^{-\infty}$, then clearly Min has a strategy giving it value $-\infty$, and we could therefore add it to $S^{-\infty}$. We will thus assume that $S^{-\infty}$ is *closed*

by attractor for Min, *i.e.* the attractor of Min towards $S^{-\infty}$ equals $S^{-\infty}$. We can define the same notion for a set $S^{+\infty}$ of states with value $+\infty$, that can be assumed closed by attractor of Max.

Lemma 7.2. *Let $S^{-\infty}$ be a set of states of value $-\infty$ closed by attractor of Min, and let $S^{+\infty}$ be a set of states of value $+\infty$ closed by attractor of Max.*

1. *Removing the states $S^{-\infty}$ from $\mathcal{G}$ will not change any other value.*
2. *Symmetrically, the states in $S^{+\infty}$ can be safely removed from $\mathcal{G}$.*

Proof of Lemma 7.2.1. Let $S' = S'_{\mathsf{Min}} \uplus S'_{\mathsf{Max}}$ denote $S \backslash S^{-\infty}$, and let $\mathcal{G}'$ denote the restriction of $\mathcal{G}$ to S', *i.e.* the game of states S', of transitions all $s \to s'$ such that $s, s' \in S'$, with state partition, labels and weights naturally inherited from $\mathcal{G}$. Let us show that for all $s \in S'$, $\mathsf{Val}_{\mathcal{G}}(s) = \mathsf{Val}_{\mathcal{G}'}(s)$.

If σ_{Min} is a strategy of Min in $\mathcal{G}$, and σ'_{Min} is a strategy of Min in $\mathcal{G}'$, such that for all plays ρ in $\mathsf{FPlays}^{\mathsf{Min}}_{\mathcal{G}'}$ $\sigma_{\mathsf{Min}}(\rho) = \sigma'_{\mathsf{Min}}(\rho)$, we say that σ_{Min} coincides with σ'_{Min}. We define the same notion for strategies of Max. All strategies of Min and Max in $\mathcal{G}'$ can be extended arbitrarily to become strategies in $\mathcal{G}$ that coincide, by making the same choices on plays that stay in $\mathcal{G}'$, and making arbitrary choices otherwise. It follows that if σ_{Min} and σ'_{Min} are strategies of Min that coincide, then for any strategy σ'_{Max} of Max in $\mathcal{G}'$, there exists a corresponding strategy σ_{Max} in $\mathcal{G}$, such that the path $\mathsf{play}_{\mathcal{G}'}(s, \sigma'_{\mathsf{Min}}, \sigma'_{\mathsf{Max}})$ is equal to $\mathsf{play}_{\mathcal{G}}(s, \sigma_{\mathsf{Min}}, \sigma_{\mathsf{Max}})$, and therefore

$$\mathsf{Val}_{\mathcal{G}'}(s, \sigma'_{\mathsf{Min}}) \leqslant \mathsf{Val}_{\mathcal{G}}(s, \sigma_{\mathsf{Min}}) \,,$$

as the supremum over strategies of Max in $\mathcal{G}$ is at least equal to the one in $\mathcal{G}'$. Similarly, if σ_{Max} and σ'_{Max} are strategies of Max that coincide, then

$$\mathsf{Val}_{\mathcal{G}'}(s, \sigma'_{\mathsf{Max}}) \geqslant \mathsf{Val}_{\mathcal{G}}(s, \sigma_{\mathsf{Max}}) \,.$$

By closure by attractor of Min, any transition starting in a state of S'_{Min} must end in a state of S' (otherwise the first state would belong to the attractor of Min towards $S^{-\infty}$). Therefore, every positional strategy of Min in $\mathcal{G}$ has a corresponding strategy in $\mathcal{G}'$ that makes the same choices. This extends naturally to the (optimal) strategies of Min switching between two positional strategies $\sigma^{\star}_{\mathsf{Min}}$ and $\sigma^{\dagger}_{\mathsf{Min}}$. Then, if we consider such an optimal strategy σ_{Min} of Min in $\mathcal{G}$, and its corresponding strategy σ'_{Min} in $\mathcal{G}'$, it holds that $\mathsf{Val}_{\mathcal{G}'}(s) \leqslant \mathsf{Val}_{\mathcal{G}'}(s, \sigma'_{\mathsf{Min}})$ by definition of Val as an infimum over strategies, and $\mathsf{Val}_{\mathcal{G}'}(s, \sigma'_{\mathsf{Min}}) \leqslant \mathsf{Val}_{\mathcal{G}}(s, \sigma_{\mathsf{Min}})$ as explained before. Finally, $\mathsf{Val}_{\mathcal{G}}(s, \sigma_{\mathsf{Min}}) = \mathsf{Val}_{\mathcal{G}}(s)$ by optimality of σ_{Min}, and thus $\mathsf{Val}_{\mathcal{G}'}(s) \leqslant \mathsf{Val}_{\mathcal{G}}(s)$. If Min has no optimal strategies in $\mathcal{G}$, there exists a family of such switching strategies whose value converges towards $\mathsf{Val}_{\mathcal{G}}(s)$, and the same result holds.

From every state s of S'_{Max}, a positional strategy that chooses a transition jumping into $S^{-\infty}$ must have value $-\infty$. Any other choice would thus be equal or better, and by closure by attractor of Min, there must exist a transition starting in s that ends in S'. Therefore, there exists an optimal positional strategy σ_{Max} of Max in $\mathcal{G}$ with a corresponding strategy σ'_{Max} in $\mathcal{G}'$. Then, it holds that $\mathsf{Val}_{\mathcal{G}'}(s) \geqslant \mathsf{Val}_{\mathcal{G}'}(s, \sigma'_{\mathsf{Max}}) \geqslant \mathsf{Val}_{\mathcal{G}}(s, \sigma_{\mathsf{Max}}) = \mathsf{Val}_{\mathcal{G}}(s)$. □

An entirely symmetrical proof lets us deal with $S^{+\infty}$ in the same way:

Proof of Lemma 7.2.2. Let $S' = S'_{\mathsf{Min}} \uplus S'_{\mathsf{Max}}$ denote $S \backslash S^{+\infty}$, and let $\mathcal{G}'$ denote the restriction of $\mathcal{G}$ to S'. Let us show that for all $s \in S'$, $\mathsf{Val}_{\mathcal{G}}(s) = \mathsf{Val}_{\mathcal{G}'}(s)$.

Once again, all strategies of Min and Max in $\mathcal{G}'$ can be extended arbitrarily to become strategies in $\mathcal{G}$ that coincide, such that if σ_{Min} and σ'_{Min} are strategies of Min that coincide then $\mathsf{Val}_{\mathcal{G}'}(s, \sigma'_{\mathsf{Min}}) \leqslant \mathsf{Val}_{\mathcal{G}}(s, \sigma_{\mathsf{Min}})$, and if σ_{Max} and σ'_{Max} are strategies of Max that coincide then $\mathsf{Val}_{\mathcal{G}'}(s, \sigma'_{\mathsf{Max}}) \geqslant \mathsf{Val}_{\mathcal{G}}(s, \sigma_{\mathsf{Max}})$.

By closure by attractor of Max, any transition starting in a state of S'_{Max} must end in a state of S'. Therefore, every optimal positional strategy σ_{Max} of Max in $\mathcal{G}$ has a corresponding strategy σ'_{Max} in $\mathcal{G}'$, such that $\mathsf{Val}_{\mathcal{G}'}(s) \geqslant \mathsf{Val}_{\mathcal{G}'}(s, \sigma'_{\mathsf{Max}}) \geqslant \mathsf{Val}_{\mathcal{G}}(s, \sigma_{\mathsf{Max}}) = \mathsf{Val}_{\mathcal{G}}(s)$ holds.

From every state s of S'_{Min}, a positional strategy that chooses a transition jumping into $S^{+\infty}$ must have value $+\infty$. Any other choice would thus be equal or better, and by closure by attractor of Max, there must exist a transition starting in s that ends in S'. Therefore, if Min has an optimal strategy then there exists an optimal strategy σ_{Min} of Min in $\mathcal{G}$ with a corresponding strategy σ'_{Min} in $\mathcal{G}'$, and it holds that $\mathsf{Val}_{\mathcal{G}'}(s) \leqslant \mathsf{Val}_{\mathcal{G}'}(s, \sigma'_{\mathsf{Min}}) \leqslant \mathsf{Val}_{\mathcal{G}}(s, \sigma_{\mathsf{Min}}) = \mathsf{Val}_{\mathcal{G}}(s)$. If Min only has an optimal family of strategies, the same reasoning holds. □

Let $S_{\mathsf{t}}^{-\infty}$ (resp. $S_{\mathsf{t}}^{+\infty}$) denote the set of target states that wt_{t} maps to $-\infty$ (resp. $+\infty$).

Corollary 7.1. *If a state of $\mathcal{G}$ is in the attractor of Min towards $S_{\mathsf{t}}^{-\infty}$ (resp. in the attractor of Max towards $S_{\mathsf{t}}^{+\infty}$), then it must have value $-\infty$ (resp. $+\infty$). Moreover, if we remove those states from $\mathcal{G}$, the value of the other states does not change.*

Proof. If a state s is in the attractor of Min towards $S_{\mathsf{t}}^{-\infty}$, then clearly Min has a (positional) strategy giving value $-\infty$ to s, and $\mathsf{Val}(s) = -\infty$. An attractor of Min is always closed by attractor of Min, so we can apply Lemma 7.2 and conclude. Once again, a symmetrical proof lets us deal with the attractor of Max towards $S_{\mathsf{t}}^{+\infty}$. □

Then, assuming that all final weights are finite, states with value $+\infty$ are those from which Min cannot reach the target states: thus, they can also be computed and then removed using the attractor algorithm.

Proposition 7.1. *If $\mathsf{wt}_{\mathsf{t}}(s_{\mathsf{t}}) \in \mathbb{Z}$ for all $s_{\mathsf{t}} \in S_{\mathsf{t}}$, then a state s has value $+\infty$ if and only if it is not in the attractor of Min towards S_{t}. Moreover, if we remove those states from $\mathcal{G}$, the value of the other states does not change.*

Proof. If a state s is not in the attractor of Min towards S_{t}, then Max has a (safety) strategy giving value $+\infty$ to s, and $\mathsf{Val}(s) = +\infty$. Conversely, if a state s is in the attractor of Min towards S_{t}, then Min has a strategy giving finite value to s, and $\mathsf{Val}(s) < +\infty$. We conclude by Lemma 7.2. □

We can therefore assume without loss of generality that all states have value in $\mathbb{Z} \cup \{-\infty\}$ and that all target states have final weight in $\mathbb{Z}$, since all of these attractors can be computed in linear time. As previously explained, finding all states of value

$-\infty$ is harder, but whenever we do manage to find them we can safely remove them by Lemma 7.2.

Remark. We showed that removing those states left values unchanged, but it does reduce the space of optimal strategies: some optimal strategies are no longer possible in the smaller game. We argue that this is inconsequential, as the optimal strategies of the sub-game can be extended into optimal strategies of the full game by using simple positional strategies derived from the attractor computations for the states that were removed.

7.3. Divergent weighted games

Our first contribution is to solve in polynomial time the value problem, for a subclass of finite weighted games that we call *divergent*. To the best of our knowledge, this is the first attempt to solve a non-trivial class of weighted games with arbitrary weights in polynomial time. Moreover, the same core technique is used for the decidability result in the timed setting that we will present in Chapter 10. Let us first define the class of divergent weighted games:

Definition 7.2. A weighted game $\mathcal{G}$ is divergent if every cycle ρ of $\mathcal{G}$ satisfies $\mathsf{wt}_\Sigma(\rho) \neq 0$.

Divergence is a property of the underlying weighted transition system, independent from the repartition of states between players. The term *divergent* reflects that cycling in the game ultimately makes the accumulated weight grow in absolute value. We will first formalise this intuition by analysing the strongly connected components (SCC) of the underlying transition system (the repartition of states into players does not matter for the SCC decomposition). Based on this analysis, we will obtain the following results:

Theorem 7.1. *The value problem over finite divergent weighted games is* PTIME*-complete. Moreover, deciding if a given finite weighted game is divergent is an* NL*-complete problem when weights are encoded in unary, and is in* PTIME *when they are encoded in binary.*

7.3.1. SCC analysis

Let $\mathcal{G}$ be a weighted game. A play ρ in $\mathcal{G}$ is said to be positive (resp. negative) if $\mathsf{wt}_\Sigma(\rho) > 0$ (resp. $\mathsf{wt}_\Sigma(\rho) < 0$). It follows that a cycle in a divergent weighted game is either positive or negative. Recall that a cycle is said to be *simple* if no states are visited twice (except for the common state at the beginning and the end of the cycle). An SCC of $\mathcal{G}$ refers to a set of states that are strongly connected—*i.e.* there is a path linking every pair of states in the SCC—, and to the sub-game induced by those states.

Definition 7.3. An SCC S is said to be *positive* (resp. *negative*) if every cycle in S is positive (resp. negative).

In a weighted game, SCCs may be neither positive nor negative: they could contain both positive and negative cycles, or cycles of weight 0. In contrast, divergent games are exactly the weighted games where that does not happen.

Proposition 7.2. *A weighted game $\mathcal{G}$ is divergent if and only if each SCC of $\mathcal{G}$ is either positive or negative.*

Proof. Let us first suppose that $\mathcal{G}$ is divergent. By contradiction, consider a negative cycle ρ (of cumulated weight $-p < 0$) and a positive cycle ρ' (of cumulated weight $p' > 0$) in the same SCC. Let s and s' be respectively the first states of ρ and ρ'. By strong connectivity, there exists a finite play η from s to s' and a finite play η' from s' to s. Let us consider the cycle ρ'' obtained as the concatenation of η and η'. If ρ'' has cumulated weight $q > 0$, the cycle obtained by concatenating q times ρ and p times ρ'' has cumulated weight 0, which contradicts the divergence of $\mathcal{G}$. The same reasoning on ρ'' and ρ' proves that ρ'' cannot be negative. Thus, ρ'' is a cycle of cumulated weight 0, which again contradicts the hypothesis.

Reciprocally, consider a cycle of $\mathcal{G}$. It belongs entirely to a single SCC, and must either be positive or negative. Thus, $\mathcal{G}$ is divergent. □

7.3.2. Computing values in polynomial time

Consider a divergent weighted game $\mathcal{G}$. We assume, as explained in Section 7.2.3 that all final weights are finite and that all values are in $\mathbb{Z} \cup \{-\infty\}$.

We will rely on the value iteration algorithm to compute the value of every state. Value iteration algorithms usually benefit from decomposing a game into SCCs (in polynomial time), considering them in a bottom-up fashion: starting with target states whose value is wt_t, SCCs are then considered in inverse topological order since the values of states in an SCC only depend on values of states of greater SCCs (in topological order), that have been previously computed.

Formally, each SCC S is solved individually: we remove from $\mathcal{G}$ every state that does not belong to S, except the outgoing neighbours–*i.e.* the states s' such that $s \xrightarrow{a} s'$ with $s \in S$ and $s' \notin S$. Such states s' are made targets, and their final weight is their value (that has been previously computed since they belong to a greater SCC).

Example 7.1. Consider the weighted game of Figure 7.1, where Min states are drawn with circles, and Max states with squares. Vertex s_t is the only target. Near each state is placed its value, near each transition is placed its weight when it differs from 0. For a given vector $\boldsymbol{V}$, we have for instance

$$\mathcal{F}(\boldsymbol{V})_{s_8} = \min(0 + \boldsymbol{V}_{s_t}, -1 + \boldsymbol{V}_{s_9})\,, \text{ and}$$

$$\mathcal{F}(\boldsymbol{V})_{s_2} = \max(-2 + \boldsymbol{V}_{s_1}, -1 + \boldsymbol{V}_{s_3}, -10 + \boldsymbol{V}_{s_5})\,.$$

By a computation of the attractor of $\{s_t\}$ for Min, we obtain directly that s_4 and s_7 have value $+\infty$. The inverse topological order on SCCs prescribes then to compute first the values for the SCC $\{s_8, s_9\}$, with target state s_t associated with final weight 0. Then,

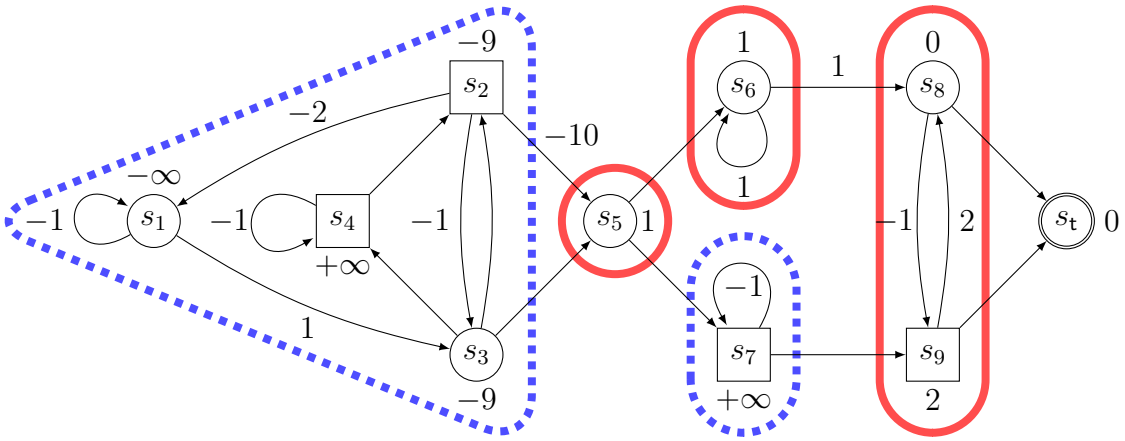


Figure 7.1.: SCC decomposition of a divergent weighted game: $\{s_1, s_2, s_3, s_4\}$ and $\{s_7\}$ are negative SCCs, $\{s_6\}$ and $\{s_8, s_9\}$ are positive SCCs, and $\{s_5\}$ is a trivial positive SCC.

we continue with SCC $\{s_6\}$, also keeping a new target state s_8 with (already computed) final weight 0. For the trivial SCC $\{s_5\}$, a single application of $\mathcal{F}$ suffices to compute the value. Finally, for the SCC $\{s_1, s_2, s_3, s_4\}$, we use a target state s_5 with final weight 1. Notice that this game is divergent, since, in each SCC, all cycles have the same sign.

For a divergent game $\mathcal{G}$, Proposition 7.2 allows us to know in polynomial time if a given SCC is positive or negative, *i.e.* if all cycles it contains are positive or negative, respectively: it suffices to consider an arbitrary cycle of it, and compute its cumulated weight. A trivial SCC (*i.e.* with a single state and no transitions) will be arbitrarily considered positive. We now explain how to compute in polynomial time the value of all states in a positive or negative SCC.

First, in case of a strongly connected component with positive cycles, we show that:

Proposition 7.3. *The value iteration algorithm applied on a positive SCC with q states stabilises after at most q steps.*

This is a corollary of Lemma 7.1. We also provide an independent proof, that will be generalisable to the timed setting.

Proof (inspired by techniques used in [BCFL04]). There are no negative cycles in the SCC, thus there are no states with value $-\infty$ in the SCC, and all values are finite. Let K be an upper bound on the values $|\boldsymbol{V}_s^q|$ obtained after q steps of the algorithm.[5] Fix an integer $p > (2K + w_{\max}(q-1))q$. We will show that the values obtained after $q+p$ steps are identical to those obtained after q steps only. Therefore, since the algorithm computes non-increasing sequences of values, we have indeed stabilised after q steps only.

[5] After q steps, the value iteration algorithm has set to a finite value all states, since it extends the attractor computation.

Assume the existence of a state s such that $\boldsymbol{V}_s^{q+p} < \boldsymbol{V}_s^q$. By induction on p, we show the existence of a state s' and a finite play ρ from s to s' with length p and cumulated weight $\boldsymbol{V}_s^{q+p} - \boldsymbol{V}_{s'}^q$: the play is composed of the transitions that optimise successively the min/max operator in $\mathcal{F}$.

Claim. *For all $i < j \in \mathbb{N}$, if $\boldsymbol{V}^j \neq \boldsymbol{V}^i$ then for all $s \in S$ there exist s' and a play ρ from s to s' with $|\rho| = j - i$ and $\mathsf{wt}_\Sigma(\rho) = \boldsymbol{V}_s^j - \boldsymbol{V}_{s'}^i$.*

Proof of Claim. Let us fix i, and prove it by induction on $j > i$.

Initialisation: If $j = i + 1$, we applied one step of the value iteration algorithm between $\boldsymbol{V}^i$ and $\boldsymbol{V}^j$, so for all s there exist s' and a transition $s \xrightarrow{a} s'$ such that $\mathsf{wt}(s, a, s') = \boldsymbol{V}_s^{i+1} - \boldsymbol{V}_{s'}^i$.

Iteration: We assume the property holds for $j - 1 > i$, and $\boldsymbol{V}^j \neq \boldsymbol{V}^i$. We applied one step of the value iteration algorithm between $\boldsymbol{V}^{j-1}$ and $\boldsymbol{V}^j$, so for all s there exist s' and a transition $s \xrightarrow{a} s'$ such that $\mathsf{wt}(s, a, s') = \boldsymbol{V}_s^j - \boldsymbol{V}_{s'}^{j-1}$. We apply the property on i and $j - 1$ ($\boldsymbol{V}^{j-1} \neq \boldsymbol{V}^i$ because $\boldsymbol{V}^j \neq \boldsymbol{V}^i$ and as soon as $\boldsymbol{V}$ stabilises, the fixpoint is reached and the iteration stops), and obtain that for all $s' \in S$ there exist s'' and a play ρ from s' to s'' with $|\rho| = j - 1 - i$ and $\mathsf{wt}_\Sigma(\rho) = \boldsymbol{V}_{s'}^{j-1} - \boldsymbol{V}_{s''}^i$. Then we define $\rho' = s \xrightarrow{a} s' \xrightarrow{\rho} s''$ and it holds that $|\rho'| = j - i$ and $\mathsf{wt}_\Sigma(\rho') = \boldsymbol{V}_s^j - \boldsymbol{V}_{s''}^i$. △

This finite play being of length greater than $(2K + w_{\max}(q-1))q$, there is at least one state appearing more than $2K + w_{\max}(q-1)$ times. Thus, it can be decomposed into at least $2K + w_{\max}(q-1)$ cycles and a finite play ρ' visiting each state at most once. The cumulate weight of ρ' is then at least $-(q-1)w_{\max}$, and as all cycles in the SCC are positive, the cumulated weight of ρ is at least $2K + w_{\max}(q-1) - (q-1)w_{\max} = 2K$. Then, $\boldsymbol{V}_s^{q+p} - \boldsymbol{V}_{s'}^q \geqslant 2K$, so $\boldsymbol{V}_s^{q+p} \geqslant 2K + \boldsymbol{V}_{s'}^q \geqslant K$. But $K \geqslant \boldsymbol{V}_s^q$, so $\boldsymbol{V}_s^{q+p} \geqslant \boldsymbol{V}_s^q$, and that is a contradiction. □

Example 7.2. For the SCC $\{s_8, s_9\}$ of the game in Figure 7.1, starting from $\boldsymbol{V}$ mapping s_8 and s_9 to $+\infty$, and s_t to 0, after one iteration, $\boldsymbol{V}_{s_8}$ changes for value 0, and after the second iteration, $\boldsymbol{V}_{s_9}$ stabilises to value 2.

Consider then the case of a negative SCC. Contrary to the previous case, we must deal with states of value $-\infty$. However, in a negative SCC, those states are easy to find[6]. These are all states where Max cannot unilaterally guarantee to reach a target state:

Proposition 7.4. *In a negative SCC with no states of value $+\infty$, states of value $-\infty$ are all the ones not in the attractor of* Max *to the targets.*

Proof. Consider a state s in the attractor of Max to the targets. Then, if Max applies a winning positional strategy for the reachability objective to the target states, all strategies of Min will generate a play from s reaching a target after at most $|S|$ steps. This implies that s has a finite value in the game.

[6]This is in contrast with the general case of (non divergent) finite weighted games where the problem of deciding if a state has value $-\infty$ is as hard as solving mean-payoff games [BGHM16].

Reciprocally, if s is not in the attractor, by determinacy of games with reachability objectives, Min has a (positional) strategy σ_{Min} to ensure that no strategy of Max permits to reach a target state from s. Applying σ_{Min} long enough to generate many negative cycles, before switching to a strategy allowing Min to reach the target (such a strategy exists since no states have value $+\infty$ in the game), allows Min to obtain from s a negative weight as small as possible. Therefore s has value $-\infty$. □

Thus, we can compute (and remove) states of value $-\infty$ in polynomial time for a negative SCC. Then, finite values of other states can be computed in polynomial time with the following procedure. From a negative SCC $\mathcal{G}$ that has no states of value $+\infty$ or $-\infty$, consider the dual (positive) SCC $\widetilde{\mathcal{G}}$ obtained by: (i) switching states of Min and Max; (ii) taking the opposite of every weight in transitions; (iii) taking the opposite of every final weight. Sets of strategies of both players are exchanged in those two games, so that the upper value in $\mathcal{G}$ is equal to the opposite of the lower value in $\widetilde{\mathcal{G}}$, and vice versa. Since weighted games are determined, the value of $\mathcal{G}$ is the opposite of the value of $\widetilde{\mathcal{G}}$. This strongly relies on the fact that in $\widetilde{\mathcal{G}}$, we can give the reachability objective to Max instead of Min (by setting the weight of an infinite play to $-\infty$) without changing the values, because both players have attractor strategies towards target states. Then, the value of $\mathcal{G}$ can be deduced from the value of $\widetilde{\mathcal{G}}$, for which Proposition 7.3 applies. We may also interpret this result as follows:

Proposition 7.5. *The value iteration algorithm, initialised with $\boldsymbol{V}_v^0 = -\infty$ (for all s), applied on a negative SCC with q states, and no states of value $+\infty$ or $-\infty$, stabilises after at most q steps.*

Proof. It is immediate that the vectors computed with this modified value iteration (that computes the smallest fixpoint of $\mathcal{F}$) are exactly the opposite vectors of the ones computed in the dual positive SCC. The previous explanation is then a justification of the result. □

Example 7.3. Consider the SCC $\{s_1, s_2, s_3, s_4\}$ of the game in Figure 7.1, where the value of state s_5 has been previously computed. We already know that s_4 has value $+\infty$ so we do not consider it further. The attractor of $\{s_5\}$ for Max is $\{s_2, s_3\}$, so that the value of s_1 is $-\infty$. Then, starting from $\boldsymbol{V}^0$ mapping s_2 and s_3 to $-\infty$, the value iteration algorithm computes this sequence of vectors: $\boldsymbol{V}^1 = (s_2 \mapsto -9, s_3 \mapsto -\infty)$ (Max tries to maximise the payoff, so he prefers jumping to the target to obtain $-10 + 1$ than going to s_3 where he gets $-1 - \infty$, while Min chooses s_2 to still guarantee $0 - \infty$), $\boldsymbol{V}^2 = (s_2 \mapsto -9, s_3 \mapsto -9)$ (now, Min has a choice between the target giving $0 + 1$ or s_3 giving $0 - 9$). Finally, $\boldsymbol{V}^3 = (s_2 \mapsto -9, s_3 \mapsto -9)$ and the fixpoint has been reached.

In a divergent weighted game where all values are finite, optimal strategies exist. Optimal strategies for both players can be obtained by combining optimal strategies in each SCC, the latter being obtained as explained in Section 7.2.

7.3.3. Polynomial lower bound

Let us show that the value problem is PTIME-hard. This comes from a reduction (in logarithmic space) of the problem of solving finite games with reachability objectives [Imm81]. To a reachability game, we simply set the weight of every transition to 1 and the final weight of every target to 0, making it a divergent weighted game. Then, Min wins the reachability game if and only if the value in the weighted game is lower than $|S|$.

7.3.4. Deciding divergence

Let us study the *membership problem* for divergent weighted games, *i.e.* the decision problem that asks if a given weighted game is divergent.

We will rely on the characterization of divergent games in term of SCCs given in Proposition 7.2. First, we note that simple cycles are enough to ensure that an SCC is positive (resp. negative), providing us with an efficient way to check this property:

Lemma 7.3. *An SCC S is positive (resp. negative) if and only if every simple cycle in S is positive (resp. negative). Moreover, deciding if an SCC is positive (resp. negative) is in* NL *when weights are encoded in unary, and is in* PTIME *when they are encoded in binary.*

Proof. The direct implication holds by definition. Reciprocally, let us assume that every simple cycle in S is positive (resp. negative), and prove that every cycle ρ in S is positive (resp. negative). The cycle ρ can be decomposed into simple cycles, all belonging to S. Therefore they are all positive (resp. negative). As the cumulated weight of ρ is the sum of the cumulated weights of these simple cycles, ρ must be positive (resp. negative).

As a corollary, an SCC S is positive (resp. negative) if and only if every cycle in S, of length at most $|S|$, is positive (resp. negative).

To decide if a strongly connected $\mathcal{G}$ is positive (resp. negative), we outline two procedures: one is deterministic and will provide the polynomial upper bound on time-complexity, the other will guess a logarithmic number of bits and provide NL membership.

The deterministic algorithm proceeds as follows: With Floyd-Warshall's algorithm, one can compute the shortest paths (resp. greatest paths) adjacency matrix M in cubic time, such that $M(s, s')$ contains the minimal (resp. maximal) value in $\{\mathsf{wt}(\rho) \mid \rho \text{ simple path from } s \text{ to } s'\}$. If there exists a state s such that $M(s, s) < 0$ (resp. $M(s, s) > 0$), then S is not positive (resp. not negative) as there is a negative (resp. positive) cycle. Conversely, if $M(s, s) = 0$, we know that all simple paths from s to s have non-negative (resp. non-positive) weight, but this includes the empty path, and we defined cycles as paths of non-zero length. In this case, S is positive (resp. negative) if and only if for every pair (s, s') it holds that $M(s, s') + M(s', s) > 0$ (resp. $M(s, s') + M(s', s) < 0$).

Let us now assume that weights are encoded in unary, and present a non-deterministic procedure. Then, note that a (binary) register containing integer values in $[-B, B]$, with B polynomial in $w_{\max}$ and $|S|$, requires a number of bits at most logarithmic in the size of $\mathcal{G}$. An SCC is *not* positive (resp. not negative), if and only if it contains a cycle of non-positive (resp. non-negative) cumulated weight, of length bounded by $|S|$. We

can guess such a cycle ρ on-the-fly, keeping in memory its cumulated weight (smaller than $B = w_{\max} \times |S|$ in absolute value), its initial state, and its current length, all in logarithmic space. If the length of the cycle exceeds $|S|$, the guess is invalid. Similarly, we can verify that the last state equals the first, and that the computed cumulated weight is indeed non-positive (resp. non-negative). Therefore, deciding if S is positive (resp. negative) is in $\mathsf{coNL} = \mathsf{NL}$ [Imm88, Sze88]. Note that when weights are encoded in binary this procedure only gives coNP membership. □

Let us now explain why the membership problem is an NL-complete problem when weights are encoded in unary. First, to prove the membership in NL, notice that a weighted game is *not divergent* if and only if there is a non-negative cycle and a non-positive cycle belonging to the same SCC, both of length at most $|S|$. This can be tested in NL, using a non-deterministic procedure similar to the one from Lemma 7.3. We first guess a starting state for both cycles. Verifying that those are in the same SCC can be done in NL by using standard reachability analysis. Then, we once again guess the two cycles on-the-fly, keeping in memory their accumulated weights in logarithmic space. Therefore, testing divergence is in $\mathsf{coNL} = \mathsf{NL}$ [Imm88, Sze88].

The NL-hardness (indeed coNL-hardness, which is equivalent [Imm88, Sze88]) is shown by a reduction of the reachability problem in a finite automaton. More precisely, we consider a finite automaton with a starting state and a different target state without outgoing transitions. We construct from it a weighted game by distributing all states to Min, and equipping all transitions with weight 1. We also add a loop with weight -1 on the target state and a transition from the target state to the initial state with weight 0. Then, the game is not divergent if and only if the target can be reached from the initial state in the automaton.

When weights are encoded in binary, the previous decision procedure gives NP membership. However, we can achieve a PTIME upperbound by computing the strongly connected components and then using Lemma 7.3 to check that each SCC is either positive or negative.

This concludes the proof of Theorem 7.1.

7.4. Almost-divergent weighted games

With divergent weighted games, we described a class where the value problem is polynomial instead of pseudo-polynomial. This gain in complexity came at a cost: the absence of cycles of weight 0. In this section, we argue that some of those cycles can be allowed: we will extend the divergent class to games that contain cycles of weight 0 under a stability by decomposition requirement, while keeping a polynomial complexity.

If ρ is a cycle $s_0 \xrightarrow{a_1} s_1 \dots s_{n-1} \xrightarrow{a_n} s_0$, it is either simple (*i.e.* for all i, j such that $0 \leqslant i < j < n$, $s_i \neq s_j$), or we can extract smaller cycles from it. Indeed, if ρ is not simple, there exists a pair (i, j) such that $0 \leqslant i < j < n$ and $s_i = s_j$. Then, for such a pair, we can write $\rho = \rho_1\rho_2\rho_3$ such that $|\rho_1| = i$, $|\rho_3| = n - j$. It follows that ρ_2 is a cycle around s_i and $\rho_1\rho_3$ is a cycle around s_0. This process is called a decomposition of ρ into

smaller cycles $\rho' = \rho_1\rho_3$ and $\rho'' = \rho_2$, with $\mathsf{wt}_\Sigma(\rho) = \mathsf{wt}_\Sigma(\rho') + \mathsf{wt}_\Sigma(\rho'')$. As there could exist several pairs (i, j) such that $s_i = s_j$, there could exist multiple decompositions of ρ into smaller cycles. A cycle ρ in $\mathcal{G}$ is called a 0-cycle if $\mathsf{wt}_\Sigma(\rho) = 0$. Let us now define the class of almost-divergent weighted games:

Definition 7.4. A weighted game $\mathcal{G}$ is *almost-divergent* if every 0-cycle ρ of $\mathcal{G}$ satisfies the following property: for every decomposition of ρ into smaller cycles ρ' and ρ'', ρ' and ρ'' are 0-cycles.

Intuitively, a game is almost-divergent if its cycles of weight different from 0 cannot be combined to create a 0-cycle. Almost-divergence is a weaker property than divergence, and thus every divergent weighted game is almost-divergent. We start by analysing the strongly connected components of $\mathcal{G}$, then derive the following results, extending Theorem 7.1:

Theorem 7.2. *The value problem over finite almost-divergent weighted games is* PTIME*-complete. Moreover, deciding if a given finite weighted game is almost-divergent is an* NL*-complete problem when weights are encoded in unary, and is in* PTIME *when they are encoded in binary.*

7.4.1. SCC analysis

A play ρ in $\mathcal{G}$ is said to be non-negative (resp. non-positive) if $\mathsf{wt}_\Sigma(\rho) \geqslant 0$ (resp. $\mathsf{wt}_\Sigma(\rho) \leqslant 0$). An SCC S is said to be *non-negative* (resp. *non-positive*) if every cycle in S is non-negative (resp. non-positive). We prove the following characterisation of almost-divergent games in terms of SCCs, extending Proposition 7.2.

Proposition 7.6. *A weighted game $\mathcal{G}$ is almost-divergent if and only if each SCC of $\mathcal{G}$ is either non-negative, or non-positive.*

Proof. Let us first suppose that $\mathcal{G}$ is almost-divergent. By contradiction, consider a negative simple cycle ρ (of weight $-p < 0$) and a positive simple cycle ρ' (of weight $p' > 0$) in the same SCC. Let s and s' be respectively the first states of ρ and ρ'. By strong connectivity, there exists a finite play η from s to s' and a finite play η' from s' to s. Let us consider the cycle $\rho'' = \eta\eta'$. If ρ'' has weight $q > 0$, the cycle obtained by concatenating q times ρ and p times ρ'' has weight 0. However, there exists a decomposition of $(\rho)^q(\rho'')^p$ into smaller cycles that obtains ρ as one of the smaller cycles, which contradicts the almost-divergence of $\mathcal{G}$ as $\mathsf{wt}_\Sigma(\rho) < 0$. The same reasoning on ρ'' and ρ' proves that ρ'' cannot be negative. Thus, ρ'' is a 0-cycle. Then, $(\rho)^{p'}\eta(\rho')^p\eta'$ is a 0-cycle, which again contradicts the hypothesis as one of its decompositions produces ρ.

Reciprocally, consider a cycle of $\mathcal{G}$ of cumulated weight 0, and one of its decomposition into smaller cycles. Both smaller cycles belong to the same SCC, therefore they are both non-negative or both non-positive. As the accumulated weight of the cycle is the sum of the weights of these smaller cycles they must both be 0-cycles. Therefore, $\mathcal{G}$ is almost-divergent. □

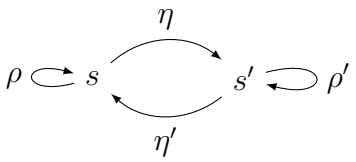


Figure 7.2.: Proof scheme of Proposition 7.6.

7.4.2. Kernel of an almost-divergent weighted game

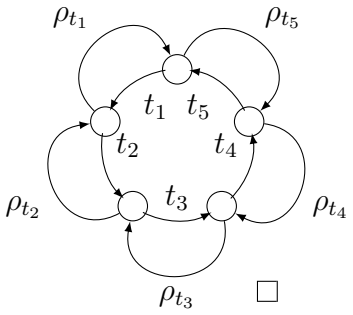
We define the kernel K as a subgraph of $\mathcal{G}$ containing all 0-cycles, such that every 0-cycle is in K and every cycle in K is a 0-cycle. Such sub-graph may not exist in general, but we show that it is always the case in almost-divergent games.

Let T_{K} be the set of transitions of $\mathcal{G}$ belonging to a *simple* 0-cycle, and S_{K} be the set of states covered by T_{K}. We define the kernel K of $\mathcal{G}$ as the subgame of $\mathcal{G}$ defined by S_{K} and T_{K}. Transitions in $T \backslash T_{\mathsf{K}}$ with starting state in S_{K} are called the output transitions of K. We define it using only simple 0-cycles in order to ensure its computability (in polynomial time). However, we now show that this is of no harm, since the kernel contains exactly all the 0-cycles.

Proposition 7.7. *A cycle of $\mathcal{G}$ is entirely in K if and only if it is a 0-cycle.*

Proof. As $\mathcal{G}$ is almost-divergent, the decomposition of every 0-cycle into simple cycles is a set of 0-cycles in K. Thus, every 0-cycle is in K.

We now prove that every cycle in K is a 0-cycle. By construction, every transition $t \in T_{\mathsf{K}}$ is part of a simple 0-cycle. Thus, to every transition $t \in T_{\mathsf{K}}$, we can associate a play ρ_t such that $t\rho_t$ is a simple 0-cycle. Then, observe that if $t_1 \cdots t_n$ is a finite play in K, then $t_1 t_2 \cdots t_n \rho_{t_n} \cdots \rho_{t_2} \rho_{t_1}$ is a 0-cycle of $\mathcal{G}$. We showed that if $\rho = t_1 \cdots t_n$ is a cycle of $\mathcal{G}$ in K, then there exists a cycle $\rho' = \rho_{t_n} \cdots \rho_{t_2} \rho_{t_1}$ such that $\rho\rho'$ is a 0-cycle, therefore ρ is a 0-cycle by almost-divergence.



□

Using K we can compute the states with value $-\infty$:

Lemma 7.4. *In an SCC of $\mathcal{G}$, the set of states with value $-\infty$ is computable in time linear in the number of states in $\mathcal{G}$. These states can be removed without changing any other value.*

Proof. If the SCC is non-negative, the cumulated weight cannot decrease along a cycle, and there can be no state of value $-\infty$ (as final weights are finite).

If the SCC is non-positive, we let T_{t} be the set of transitions of $\mathcal{G}$ whose end state belongs to S_{t}. Notice that the kernel cannot contain target states since targets do not have outgoing edges. We will prove that a configuration has value $-\infty$ if and only if it belongs to a state where player Min can ensure the LTL formula on transitions:

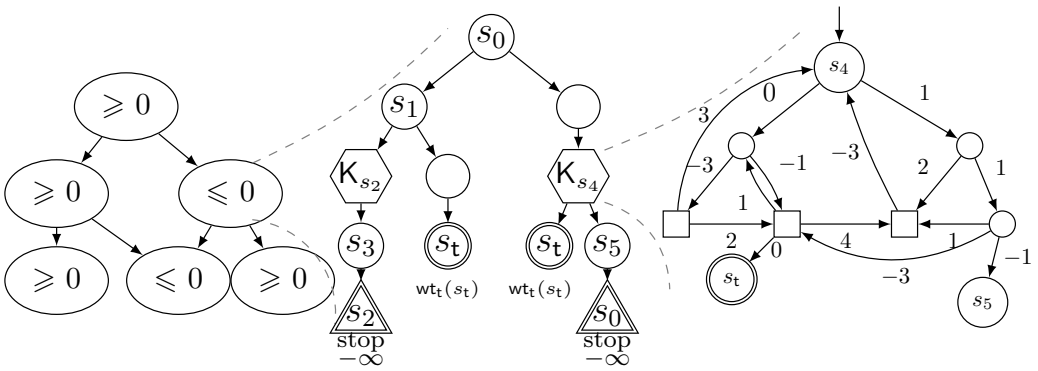


Figure 7.3.: SCC decomposition of $\mathcal{G}$, semi-unfolding of an SCC, kernel

$\phi = (\mathrm{G}\,\neg T_\mathsf{t}) \wedge \neg \mathrm{FG}\,T_\mathsf{K}$. The procedure to detect $-\infty$ states will thus consist of three attractor computations, one for each LTL operator, and can be done in time linear in $|S|$.

Since ω-regular games are determined, this is equivalent to saying that a configuration has finite value if and only if it belongs to a state where Max can ensure $\neg\phi = (\mathrm{F}T_\mathsf{t}) \vee \mathrm{FG}T_\mathsf{K}$. If s is a state where Min can ensure ϕ, he can ensure a value of $-\infty$ by avoiding S_t for as long as he desires, while not getting stuck in K, and thus going through an unbounded number of negative cycles by Proposition 7.7. This proves that a state where Max cannot ensure $\neg\phi$ contains only valuations of value $-\infty$. Conversely, if s is a state where Max can ensure $\neg\phi$, then from s, Max must be able to enforce either reaching S_t or staying in K forever. In both cases, Max can ensure a value above $-\infty$.

Finally, the set of $-\infty$ states is closed by attractor of Min, therefore by Lemma 7.2 they can be safely removed. □

We will assume from now on that every state of $\mathcal{G}$ has value in $\mathbb{Z}$.

7.4.3. Semi-unfolding

Let us prove that the value problem is PTIME-complete on almost-divergent weighted games. We derive PTIME-hardness from the sub-class of divergent games. In order to obtain an upper bound in PTIME, we will again compute the value of states in $\mathcal{G}$ in an SCC by SCC fashion, following the inverse topological order of the SCC decomposition. For non-negative SCCs, Lemma 7.1 can be used to compute values. However, this does not hold on non-positive SCCs, as they contain negative cycles. We will give a procedure that can handle both cases.

For an SCC of $\mathcal{G}$ and an initial state s_0 provided by the SCC decomposition, we show that the game on the SCC is equivalent to a game with a tree-shaped structure built as a semi-unfolding of $\mathcal{G}$ from s_0, with certain nodes of the tree being *kernels*. These kernels are strongly connected parts of $\mathcal{G}$ included in K that contain all cycles of weight 0. The semi-unfolding is stopped either when reaching a target state, or when the depth reaches $|S \backslash S_\mathsf{t}|$.

Given an almost-divergent game $\mathcal{G}$, we will describe the construction of its *semi-unfolding* $\mathcal{T}(\mathcal{G})$.

If s is in K, we let K_s be the part of K accessible from s (note that K_s is strongly connected as K is a union of cycles). We define the output transitions of K_s as being the output transitions of K accessible from s. If s is not in K, we define the output transitions of s as the transitions of $\mathcal{G}$ that start in s.

We define a tree T whose nodes will either be labelled by states $s \in S$ or by kernels K_s, and whose edges will be labelled by output transitions in $\mathcal{G}$. The root of the tree T is labelled with s_0, or K_{s_0} (if s_0 belongs to the kernel), and the successors of a node of T are then recursively defined by its output transitions. When a state s is reached by an output transition, the child is labelled by K_s if $s \in \mathsf{K}$, otherwise it is labelled by s. Edges in T are labelled by the transitions used to create them. Along every branch, we stop the construction when either a final state is reached (*i.e.* a state not inside the current SCC) or the branch has length $|S \backslash S_\mathsf{t}|$. As a corollary, every branch either ends in a target state or contains two nodes labelled by the same state (s or K_s). Leaves of T with a state belonging to S_t are called *target leaves*, others are called *stopped leaves*. In the case where the stopped leaf of a branch is a node K_s, we label the stopped leaf by s instead.

We now transform T into a weighted game $\mathcal{T}(\mathcal{G})$, by replacing every node labelled by a state s by a different copy $\tilde{s}$ of s. Those states are said to inherit from s. Edges of T are replaced by the transitions labelling them, and have a similar notion of inheritance. Every non-leaf node labelled by a kernel K_s is replaced by a copy of the weighted game K_s, output transitions being plugged in the expected way. State partition between players and transition weights are inherited from the copied states and transitions of $\mathcal{G}$. The only initial state of $\mathcal{T}(\mathcal{G})$ is the state denoted by $\tilde{s}_0$ inherited from s_0 in the root of T (either s_0 or K_{s_0}). The target states of $\mathcal{T}(\mathcal{G})$ are the states derived from leaves of T. If $\mathcal{G}$ is a non-negative (resp. non-positive) SCC, the final weight function wt_t is inherited from $\mathcal{G}$ on target leaves and set to $+\infty$ (resp. $-\infty$) on stopped leaves.

We will now prove that $\mathcal{G}$ and $\mathcal{T}(\mathcal{G})$ are equivalent on the root state. Two plays ρ and $\tilde{\rho}$ in $\mathcal{G}$ and $\mathcal{T}(\mathcal{G})$, resp. are said to *mimic* each other if $|\rho| = |\tilde{\rho}|$ and for every $1 \leqslant i \leqslant |\rho|$ the i-th transition of $\tilde{\rho}$ is inherited from the i-th transition of ρ. Then, the plays of $\mathcal{G}$ starting in the initial state that cannot be mimicked in $\mathcal{T}(\mathcal{G})$ are not useful for value computation, which is formalised by Proposition 7.8:

Proposition 7.8. *For every strongly connected, almost-divergent weighted game $\mathcal{G}$, with an initial state s_0, $\mathsf{Val}_\mathcal{G}(s_0) = \mathsf{Val}_{\mathcal{T}(\mathcal{G})}(\tilde{s}_0)$.*

Proof. Recall that we only left finite values in $\mathcal{G}$ (in the final weight functions, in particular). We first show that the value is also finite in $\mathcal{T}(\mathcal{G})$. Indeed, if $\mathsf{Val}_{\mathcal{T}(\mathcal{G})}(\tilde{s}_0) = +\infty$, since we assumed all final weights of $\mathcal{G}$ bounded, we are necessarily in the non-negative case, and Max is able to ensure stopped leaves reachability. Then, for any positional strategy σ_Min of Min in $\mathcal{G}$, there exists a strategy σ_Max of Max ensuring that the target is never reached in $\mathsf{play}(s_0, \sigma_\mathsf{Min}, \sigma_\mathsf{Max})$. This extends to non-positional strategies of Min, as it implies that s_0 is not in the attractor of Min towards S_t, and therefore $\mathsf{Val}_\mathcal{G}(s_0) = +\infty$. Thus, we obtain a contradiction. If $\mathsf{Val}_{\mathcal{T}(\mathcal{G})}(\tilde{s}_0) = -\infty$, we are necessarily in the non-positive case, and by construction this implies having Min ensuring stopped

leaves reachability in $\mathcal{T}(\mathcal{G})$. Symmetrically from the $+\infty$ case, we obtain that s_0 must not be in the attractor of Max towards S_t, *i.e.* Min can ensure the LTL formula on transitions $\mathrm{G}\,\neg T_\mathsf{t}$. Moreover, Min can also ensure the formula $\neg\mathrm{FG}\,T_\mathsf{K}$, as otherwise Max would have a strategy preventing him from reaching a stopped leaf, by staying in a kernel forever. Then, Min can ensure the LTL formula on transitions $(\mathrm{G}\,\neg T_\mathsf{t}) \wedge \neg\mathrm{FG}\,T_\mathsf{K}$ at s_0 in $\mathcal{G}$. As observed in the proof of Lemma 7.4, this implies $\mathsf{Val}_\mathcal{G}(s_0) = -\infty$, and is a contradiction.

Then, strategies and plays of $\mathcal{T}(\mathcal{G})$ starting from $\tilde{s}_0$ can be mimicked in $\mathcal{G}$, therefore if $\mathcal{G}$ is non-negative then $\mathsf{Val}_\mathcal{G}(s_0) \leqslant \mathsf{Val}_{\mathcal{T}(\mathcal{G})}(\tilde{s}_0)$: We can fix an optimal strategy σ_Min for Min in $\mathcal{T}(\mathcal{G})$. It is a strategy of value less than $+\infty$, so every play derived from σ_Min in $\mathcal{T}(\mathcal{G})$ reaches a target leaf, and can be mimicked in $\mathcal{G}$. Therefore, σ_Min can be mimicked in $\mathcal{G}$, where it is also winning, with the same weight. If $\mathcal{G}$ is non-positive, the same reasoning applies by considering an optimal strategy for Max in $\mathcal{T}(\mathcal{G})$ (by determinacy we can take the viewpoint of Max), and gives $\mathsf{Val}_\mathcal{G}(s_0) \geqslant \mathsf{Val}_{\mathcal{T}(\mathcal{G})}(\tilde{s}_0)$.

Let us now show that $\mathsf{Val}_{\mathcal{T}(\mathcal{G})}(\tilde{s}_0) \leqslant \mathsf{Val}_\mathcal{G}(s_0)$ when $\mathcal{G}$ is non-negative. There are no negative cycles, so Min has an optimal strategy σ_Min that is positional by Lemma 7.1. Let us fix a strategy σ_Max of Max in $\mathcal{G}$, and let ρ be their outcome $\mathsf{play}_\mathcal{G}(s_0, \sigma_\mathsf{Min}, \sigma_\mathsf{Max})$. Since σ_Min is optimal and positional and stopped leaves have final weight $+\infty$, ρ must be a simple path (without cycles) that reaches a target. Then, for every σ_Max all such plays ρ can be mimicked in $\mathcal{T}(\mathcal{G})$, and $\mathsf{Val}_{\mathcal{T}(\mathcal{G})}(\tilde{s}_0) \leqslant \mathsf{Val}_{\mathcal{T}(\mathcal{G})}(\tilde{s}_0, \sigma_\mathsf{Min}) = \mathsf{Val}_\mathcal{G}(s_0)$. Once again, if $\mathcal{G}$ is non-positive, the same reasoning applies by considering an optimal positional strategy for Max in $\mathcal{G}$, and gives $\mathsf{Val}_{\mathcal{T}(\mathcal{G})}(\tilde{s}_0) \geqslant \mathsf{Val}_\mathcal{G}(s_0)$. □

In order to compute the value of a state s_0 of $\mathcal{G}$, one could construct the semi-unfolding of root s_0, and compute its value. Indeed, every cycle in $\mathcal{T}(\mathcal{G})$ belongs to K, so they must all be 0-cycles, therefore by Lemma 7.1 we can compute $\mathsf{Val}_{\mathcal{T}(\mathcal{G})}(\tilde{s}_0)$ in time polynomial in the size of $\mathcal{T}(\mathcal{G})$. However, this would be an exponential algorithm, as the number of nodes in T can be exponential in $|S|$. We argue that this exponential blow-up can be avoided: when two nodes of T are at the same depth and are labelled by the same state they can be merged, producing a graph T that is acyclic instead of tree-shaped, with at most quadratically many states. This does not change the value of the resulting weighted game $\mathcal{T}(\mathcal{G})$ at its root, because the two merged nodes had the same sub-tree, and therefore were states with the same value in $\mathcal{T}(\mathcal{G})$. This optimization on the construction of $\mathcal{T}(\mathcal{G})$ is performed on-the-fly, while the semi-unfolding is constructed, such that constructing $\mathcal{T}(\mathcal{G})$ (and solving it by Lemma 7.1) can be done in time polynomial in the size of $\mathcal{G}$.

Remark. In contrast with divergent games, for non-positive SCCs we cannot use the dual game where weights and players are switched, because Max may not be able to enforce target reachability. Consider *e.g.* a self-loop of weight 0 controlled by Min. It does not affect value computation, but it provides Max with value $+\infty$ in the dual game. If $\mathcal{T}(\mathcal{G})$ were to be a complete unfolding (that unfolds kernels as well), as opposed to a semi-unfolding, it could wrongly have value $-\infty$ at its root, reflecting this issue.

7.4.4. Deciding almost-divergence

In order to study the membership problem for almost-divergent games—that asks if a given weighted game is almost-divergent—, we will adapt techniques developed for divergent games.

Once again, simple cycles are enough to ensure that an SCC is non-negative (resp. non-positive):

Lemma 7.5. *An SCC S is non-negative (resp. non-positive) if and only if every simple cycle in S is non-negative (resp. non-positive). Moreover, deciding if an SCC is non-negative (resp. non-positive) is in* NL *when weights are encoded in unary, and is in* PTIME *when they are encoded in binary.*

Proof. The direct implication holds by definition. Reciprocally, following the proof of Lemma 7.3, every cycle ρ can be decomposed into simple cycles, all belonging to S. Therefore they are all non-negative (resp. non-positive), and ρ must be non-negative (resp. non-positive).

As a corollary, an SCC S is non-negative (resp. non-positive) if and only if every cycle in S, of length at most $|S|$, is non-negative (resp. non-positive).

To decide if an SCC is non-negative (resp. non-positive), we rely on deterministic and non-deterministic procedures similar to the ones described in Lemma 7.3. The deterministic algorithm also uses Floyd-Warshall's algorithm to compute the shortest paths (resp. greatest paths) adjacency matrix M in cubic time, such that $M(s, s')$ contains the minimal (resp. maximal) value in $\{\mathsf{wt}(\rho) \mid \rho \text{ simple path from } s \text{ to } s'\}$. Then, S is non-negative (resp. non-positive) if and only if for every state s it holds that $M(s, s) \geqslant 0$ (resp. $M(s, s) \leqslant 0$). Let us now assume that weights are encoded in unary. An SCC is *not* non-negative (resp. not non-positive), if and only if it contains a cycle of positive (resp. negative) cumulated weight, of length bounded by $|S|$. We can guess such a cycle on-the-fly, and compute its cumulated weight in logarithmic space. Therefore, deciding if S is non-negative (resp. non-positive) is in $\mathsf{coNL} = \mathsf{NL}$ [Imm88, Sze88]. □

In order to prove the membership in NL with unary weights, notice that a weighted game is *not almost-divergent* if and only if there is a positive cycle and a negative cycle, both of length at most $|S|$, and belonging to the same SCC. This can be tested in NL using the procedure from Section 7.3.4. When weights are encoded in binary, we can achieve a PTIME upperbound with the same procedure as for divergent games, by applying Lemma 7.5 on every SCC.

The NL-hardness (indeed coNL-hardness, which is equivalent [Imm88, Sze88]) is shown by a reduction of the reachability problem in a finite automaton. More precisely, we consider a finite automaton with a starting state and a different target state without outgoing transitions. We construct from it a weighted game by distributing all states to Min, and equipping all transitions with weight 0. We also add a loop with weight 1 on the initial state, one with weight -1 on the target state, and a transition from the target state to the initial state with weight 0. Then, the game is not almost-divergent if and only if the target can be reached from the initial state in the automaton.

We have finished the proof of Theorem 7.2.

8. Weighted timed games

We now turn our attention to a timed extension of the weighted games. We will first define weighted timed games, giving their semantics in terms of *infinite* weighted games.

8.1. The timed setting

Definition 8.1. A *weighted timed game* (WTG) is a tuple $\mathcal{G} = \langle L_{\mathsf{Min}}, L_{\mathsf{Max}}, \mathcal{X}, \Sigma, L_{\mathsf{t}}, E, \mathsf{wt}\rangle$ where $\langle L = L_{\mathsf{Min}} \uplus L_{\mathsf{Max}}, \mathcal{X}, \Sigma, E\rangle$ is a timed automaton whose locations are split between players Min and Max, $\mathsf{wt}\colon E \uplus L \to \mathbb{Z}$ is the weight function, associating an integer weight with each location and edge, $L_{\mathsf{t}} \subseteq L_{\mathsf{Min}}$ is a set of target locations for player Min, and $\mathsf{wt}_{\mathsf{t}}\colon L_{\mathsf{t}} \times \mathbb{R}^{\mathcal{X}}_{\geqslant 0} \to \mathbb{R}_\infty$ is a function mapping each target configuration to a final weight of $\mathbb{R}_\infty = \mathbb{R} \cup \{-\infty, +\infty\}$.

In this work, we will not use the actions in Σ that label edges, and will therefore not represent them in our formalism.[1] Once again, final weights are non-standard and are usually set to 0 in related work.

The semantics of a weighted timed game $\mathcal{G}$ is defined in terms of the infinite weighted game $[\![\mathcal{G}]\!]$ whose states are configurations (ℓ, ν) of the underlying timed automaton. Configurations are split into players according to the location ℓ. A configuration (ℓ, ν) is a target if $\ell \in L_{\mathsf{t}}$, and its final weight is $\mathsf{wt}_{\mathsf{t}}(\ell, \nu)$. The labels of $[\![\mathcal{G}]\!]$ are given by $\mathbb{R}_{\geqslant 0} \times E$ and will encode the delay that a player wants to spend in the current location, before firing a certain edge of the timed automaton. For every delay $d \in \mathbb{R}_{\geqslant 0}$, edge $e = (\ell, g, \mathcal{Y}, \ell') \in E$ and valuation ν, there is a transition $(\ell, \nu) \xrightarrow{d,e} (\ell', \nu')$ if $\nu + d \models g$ and $\nu' = (\nu + d)[\mathcal{Y} := 0]$. The weight of such a transition takes into account both discrete and continuous costs, and is given by $d \cdot \mathsf{wt}(\ell) + \mathsf{wt}(e)$.

As usual in related work [ABM04, BCFL04, BJM15], we assume that the WTGs have non-diagonal guards where all constants are integers, that all clocks are *bounded* by the greatest constant M to appear in guards, and we restrict $[\![\mathcal{G}]\!]$ to configurations in $L \times \mathbb{R}^{\mathcal{X}}_{\geqslant 0, <M}$.[2] Without loss of generality, we suppose the absence of deadlocks in $[\![\mathcal{G}]\!]$ except on target locations, *i.e.* for each location $\ell \in L \backslash L_{\mathsf{t}}$ and valuation $\nu \in \mathbb{R}^{\mathcal{X}}_{\geqslant 0, <M}$, there exist $d \in \mathbb{R}_{\geqslant 0}$ and $(\ell, g, \mathcal{Y}, \ell') \in E$ such that $(\ell, \nu) \xrightarrow{d,e} (\ell', \nu')$, and no edges start from L_{t}. We also assume that the final weight functions satisfy a property ensuring that they can be encoded in finite space: they must be piecewise affine with a finite number of

[1] Equivalently, one might assume that every edge is equipped by a unique action.

[2] Observe that this assumption is without loss of generality for (weighted) timed automata [BFH+01], but we do not know if it is for weighted timed games: in particular in the presence of negative weights, the technique of [BFH+01] can not be directly applied.

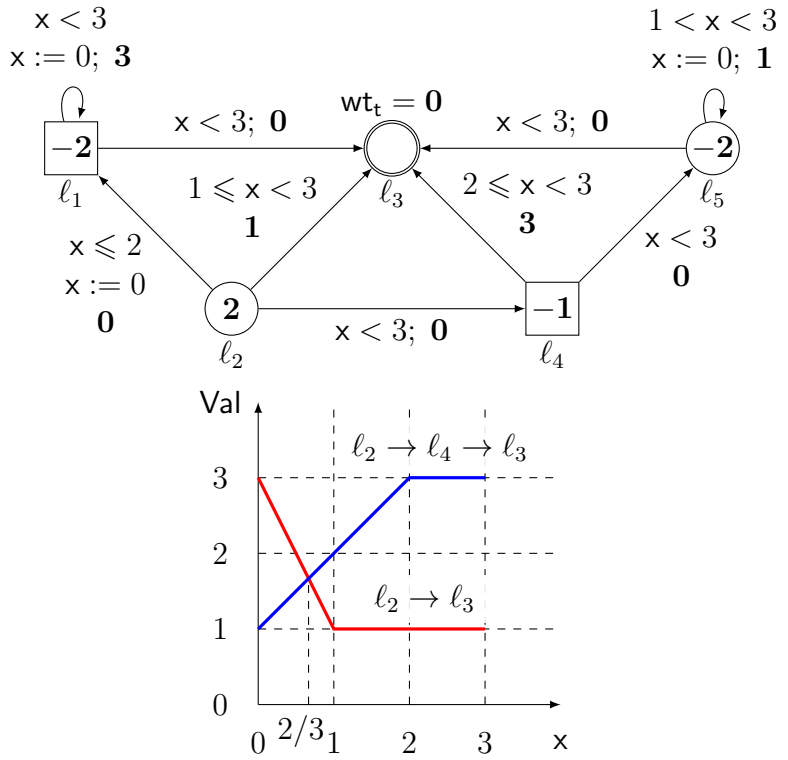


Figure 8.1.: A weighted timed game with a single clock x, and a depiction of its value function. Weights are indicated in bold font on locations and edges. Locations belonging to Min (resp. Max) are depicted by circles (resp. squares). The target location is ℓ_3, whose final weight function is zero.

pieces and are continuous on each region. In particular, infinite final weights are constant over regions, *i.e.* if some configuration (ℓ_t, ν) has final weight $+\infty$ or $-\infty$, then for every valuation ν' in the same region as ν, $\mathsf{wt}_t(\ell_t, \nu) = \mathsf{wt}_t(\ell_t, \nu')$. The standard final weight function $(\ell, \nu) \mapsto 0$ satisfies this property. Moreover, the computations we will perform in the following maintain this property as an invariant.

Example 8.1. An example of WTG satisfying those assumptions is depicted on Figure 8.1. It is easy to observe that location ℓ_1 (resp. ℓ_5) has value $+\infty$ (resp. $-\infty$). As a consequence, the value in ℓ_4 is determined by the edge to ℓ_3, and depicted in blue. In location ℓ_2, the value associated with the edge to ℓ_3 is depicted in red, and the value in ℓ_2 is obtained as the minimum of these two curves. Observe the intersection in $\mathsf{x} = 2/3$ requiring to refine the regions.

Plays, strategies, and values in the weighted timed game $\mathcal{G}$ will refer to plays, strategies, and values in $[\![\mathcal{G}]\!]$: Plays are executions in $\mathcal{G}$, strategies map non-maximal plays to a choice of delay and edge to follow, and values are defined as the best weight each player can guarantee. It is known that (turn-based) weighted timed games are determined[3],

[3]The result is stated in [BGH+15] for weighted timed games (called priced timed games) with one

i.e. $\underline{\mathsf{Val}}(\ell,\nu) = \overline{\mathsf{Val}}(\ell,\nu)$ for each location ℓ and valuation ν, therefore we use the notation Val to refer to both values. We also define $\mathsf{wt}^i(\rho)$ the weight of a maximal play ρ at horizon i, as $\mathsf{wt}(\rho)$ if ρ reaches a target in at most i steps, and $+\infty$ otherwise. Then, $\mathsf{Val}^i(s) = \inf_{\sigma_{\mathsf{Min}}} \sup_{\sigma_{\mathsf{Max}}} \mathsf{wt}^i(\mathsf{play}(s,\sigma_{\mathsf{Min}},\sigma_{\mathsf{Max}}))$ refers to the value at horizon i.

We say that a strategy $\sigma^\star_{\mathsf{Min}}$ of Min is ε-optimal if, for all (ℓ,ν), and all strategies σ_{Max} of Max,

$$\mathsf{wt}(\mathsf{play}((\ell,\nu),\sigma^\star_{\mathsf{Min}},\sigma_{\mathsf{Max}})) \leqslant \mathsf{Val}(\ell,\nu) + \varepsilon\,.$$

It is said optimal if this holds for $\varepsilon = 0$. A symmetric definition holds for optimal strategies of Max.

We denote by $w^L_{\max}$ (resp. $w^E_{\max}$) the maximal weight in absolute values of locations (resp. of edges) in $\mathcal{G}$.

$$w^L_{\max} = \max_{\ell\in L} |\mathsf{wt}(\ell)| \qquad w^E_{\max} = \max_{e\in E} |\mathsf{wt}(e)|$$

Moreover, in order to match notations from the untimed setting we denote by $w_{\max}$ a bound on the weight of transitions in $[\![\mathcal{G}]\!]$, that exists since clocks are bounded by M:

$$w_{\max} = M w^L_{\max} + w^E_{\max}$$

The integer $w_{\max}$ is at most exponential in the size of $\mathcal{G}$, and can thus be stored in polynomial space.

8.1.1. Region and corner-point abstractions

We will rely on the crucial notions of regions and their refinement with respect to a granularity $1/N$. We will also rely on the region abstraction, seen either as a finite transition system or as a timed automaton, as introduced in Chapter 2.3.3. The region abstraction of the timed automaton underlying $\mathcal{G}$ can be seen as a finite two-player turn-based game, called the region game, by assigning to player Min (resp. Max) the region states (ℓ,r) with $\ell \in L_{\mathsf{Min}}$ (resp. $\ell \in L_{\mathsf{Max}}$). Similarly, the region automaton[4] can be seen as a WTG, denoted $\mathcal{R}(\mathcal{G})$, that we will abusively call region game too. When $1/N$-regions are considered, the region game is denoted $\mathcal{R}_N(\mathcal{G})$.

By projecting away the region information of $\mathcal{R}_N(\mathcal{G})$, we simply obtain:

Lemma 8.1. *For all $\ell \in L$, $1/N$-regions r, and $\nu \in r$, $\mathsf{Val}_{\mathcal{G}}(\ell,\nu) = \mathsf{Val}_{\mathcal{R}_N(\mathcal{G})}((\ell,r),\nu)$.*

On top of regions, we will need the corner-point abstraction techniques introduced in [BBL08]. Recall that a valuation v is said to be a corner of a $1/N$-region r, if it belongs to the topological closure $\overline{r}$ and has coordinates multiple of $1/N$ ($\mathsf{v} \in (1/N)\mathbb{N}^{\mathcal{X}}$). We call corner state a triple (ℓ,r,v) that contains information about a region state (ℓ,r) of $\mathcal{R}_N(\mathcal{G})$, and a corner v of the $1/N$-region r. Every region has at most $|\mathcal{X}|+1$ corners. We now define the corner-point abstraction $\Gamma_N(\mathcal{G})$ of a WTG $\mathcal{G}$ as the WTG

clock, but the proof does not use the assumption on the number of clocks.

[4] Recall that we defined it as a timed automaton in Chapter 2.3.3.

obtained as a refinement of $\mathcal{R}_N(\mathcal{G})$ where guards on edges are enforced to stay on one of the corners of the current $1/N$-region: the locations of $\Gamma_N(\mathcal{G})$ are all corner states of $\mathcal{R}_N(\mathcal{G})$, associated to each player accordingly, and edges are all $(\ell, r, \mathsf{v}) \xrightarrow{g'', \mathcal{Y}} (\ell', r', \mathsf{v}')$ such that there exists $t = (\ell, r) \xrightarrow{g, \mathcal{Y}} (\ell', r')$ an edge of $\mathcal{R}_N(\mathcal{G})$ such that the model of guard g'' is a corner v'' satisfying the guard $\overline{g}$ (recall that $\overline{g}$ is the closed version of g), $\mathsf{v}'' \in \mathsf{PostTime}(\mathsf{v})$, $\mathsf{v}' = \mathsf{v}''[\mathcal{Y} := 0]$, and there exist two valuations $\nu \in r$, $\nu' \in r'$ such that $((\ell, r), \nu) \xrightarrow{d', t} ((\ell', r'), \nu')$ for some $d' \in \mathbb{R}_{\geqslant 0}$ (the latter condition ensures that the edge between corners is not spurious, *i.e.* created by the closure of guards).

Because of this closure operation, we must also define properly the final weight function: we simply define it over the only valuation v reachable in location (ℓ, r, v) (with $\ell \in L_{\mathsf{t}}$) by $\mathsf{wt}_{\mathsf{t}}((\ell, r, \mathsf{v}), \mathsf{v}) = \lim_{\nu \to \mathsf{v}, \nu \in r} \mathsf{wt}_{\mathsf{t}}(\ell, \nu)$ (the limit is well defined since wt_{t} is piecewise affine with a finite number of pieces on region r).

The WTG $\Gamma_N(\mathcal{G})$ can be seen as a finite weighted game (which means that there are only weights on edges), by removing guards, resets and rates of locations, and replacing the weights of edges by the actual weight of jumping from one corner to another: an edge $((\ell, r), \mathsf{v}) \xrightarrow{g'', \mathcal{Y}} ((\ell', r'), \mathsf{v}')$ becomes a transition from $((\ell, r), \mathsf{v})$ to $((\ell', r'), \mathsf{v}')$ with weight $d \cdot \mathsf{wt}(\ell) + \mathsf{wt}(t)$, with $d \in \mathbb{R}_{\geqslant 0}$ the only delay such that $[\![g'']\!] = \{\mathsf{v} + d\}$. Note that delay d is necessarily a rational of the form α/N with $\alpha \in \mathbb{N}$, since it must relate corners of $1/N$-regions. In particular, this proves that the cumulated weight $\mathsf{wt}_\Sigma(\overline{\rho})$ of a finite play $\overline{\rho}$ in $\Gamma_N(\mathcal{G})$ is indeed a rational number with denominator N.

We will call *corner play* every play $\overline{\rho}$ in the corner-point abstraction $\Gamma_N(\mathcal{G})$: it can also be interpreted as an execution in $\mathcal{G}$ where all guards are closed (as explained in the definition above). It straightforwardly projects on a finite path p in the region game $\mathcal{R}_N(\mathcal{G})$: in this case, we say again that $\overline{\rho}$ follows p. Figure 8.2 depicts a play, its projected path in the region game and one of its associated corner plays.

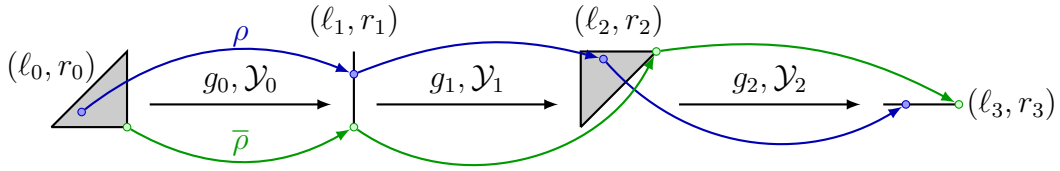


Figure 8.2.: A play ρ (in blue), its projected path p in the region game (in black), and one of its associated corner plays $\overline{\rho}$ (in green).

Let $\overline{\rho}$ be a corner play following a region path p. The weight of $\overline{\rho}$ refers to its weight in $\Gamma_N(\mathcal{G})$. Following [BBL08, Prop. 5], it is possible to find a play ρ following p close to $\overline{\rho}$, in the sense that we control the difference between their respective cumulated weights:

Lemma 8.2. *For all $\varepsilon > 0$, all region paths p, and all corner plays $\overline{\rho}$ following p, there exists a play ρ in $\mathcal{G}$ following p such that $|\mathsf{wt}_\Sigma(\rho) - \mathsf{wt}_\Sigma(\overline{\rho})| \leqslant \varepsilon$.*

Proof. If ν is a valuation and $\varepsilon > 0$, let $\mathcal{B}_\infty(\nu, \varepsilon)$ denote the open ball of radius ε centered

into ν for the infinity norm $\|.\|_\infty$ over $\mathbb{R}^{\mathcal{X}}_{\geqslant 0}$:

$$\|\nu - \nu'\|_\infty = \max_{x \in \mathcal{X}} |\nu(x) - \nu'(x)| \,.$$

We denote by $d(\rho, \overline{\rho})$ the distance between those two plays, defined as the sum of the differences in absolute value between the delays on the transitions of ρ and $\overline{\rho}$. By triangular inequality, we obtain

$$|\mathsf{wt}_\Sigma(\rho) - \mathsf{wt}_\Sigma(\overline{\rho})| \leqslant w^L_{\max} d(\rho, \overline{\rho}) \,,$$

since the same transitions are fired in ρ and $\overline{\rho}$, with only different delays.

We will now prove the following:

Claim. *For all $\varepsilon > 0$, all region paths p from r_0 to r_1, and all $\nu \in r_0 \cap \mathcal{B}_\infty(\mathsf{v}, \varepsilon)$, there exists $\nu' \in r_1 \cap \mathcal{B}_\infty(\mathsf{v}', \varepsilon)$ and ρ a play in $\mathcal{G}$ from ν to ν' following p such that $|\mathsf{wt}(\rho) - \mathsf{wt}(\overline{\rho})| \leqslant 2\varepsilon |\mathsf{p}| w^L_{\max}$.*

Proof of Claim. By the previous explanation, it is sufficient to find a play ρ such that $d(\rho, \overline{\rho}) \leqslant 2\varepsilon|\mathsf{p}|$. By induction, it is sufficient to prove a similar result only for a single edge $(\ell, r) \xrightarrow{r'', e} (\ell', r')$ of $\mathcal{R}(\mathcal{G})$, between regions r and r', with $e = \ell \xrightarrow{g, \mathcal{Y}} \ell'$. We thus consider a corner play $(\ell, \mathsf{v}) \xrightarrow{d, e} (\ell', \mathsf{v}')$ in the closed timed game $\overline{\mathcal{G}}$ from a corner $\mathsf{v} \in \overline{r}$ to a corner $\mathsf{v}' \in \overline{r'}$. Consider a valuation $\nu \in r \cap \mathcal{B}_\infty(\mathsf{v}, \varepsilon)$. We now explain how to construct a valuation $\nu' \in r' \cap \mathcal{B}_\infty(\mathsf{v}', \varepsilon)$ and $d' \geqslant 0$ such that $(\ell, \nu) \xrightarrow{d', e} (\ell', \nu')$ is a valid play in $\mathcal{G}$ and $|d - d'| \leqslant 2\varepsilon$, which implies the claim.

Let r'' be a time successor region of r such that $r''[\mathcal{Y} := 0] = r'$. We let $\mathsf{v}'' = \mathsf{v} + d$ be the corner of r'' such that $\mathsf{v}' = \mathsf{v}''[\mathcal{Y} := 0]$. Then, the timed successors of ν, *i.e.* the affine line $\nu + (1, 1, \ldots, 1)\mathbb{R}$, intersect the set $r'' \cap \mathcal{B}_\infty(\mathsf{v}'', \varepsilon)$ in a valuation ν'': indeed, lines obtained by time elapsing starting from ν and v are parallel, and r'' is a time successor of r. There exists d' such that $\nu'' = \nu + d'$. Moreover,

$$d' = \|\nu - \nu''\|_\infty \leqslant \|\nu - \mathsf{v}\|_\infty + \|\mathsf{v} - \mathsf{v}''\|_\infty + \|\mathsf{v}'' - \nu''\|_\infty \leqslant 2\varepsilon + d \,, \text{ and}$$

$$d = \|\mathsf{v} - \mathsf{v}'\|_\infty \leqslant \|\mathsf{v} - \nu\|_\infty + \|\nu - \nu'\|_\infty + \|\nu' - \mathsf{v}'\|_\infty \leqslant 2\varepsilon + d' \,,$$

so that $|d - d'| \leqslant 2\varepsilon$. Letting $\nu' = \nu''[\mathcal{Y} := 0]$, we have $(\ell, \nu) \xrightarrow{d', e} (\ell', \nu')$ and $\nu' \in r' \cap \mathcal{B}_\infty(\mathsf{v}', \varepsilon)$. △

This claim implies Lemma 8.2, and concludes our proof. □

Thus, corner plays allow one to obtain faithful information on the plays that follow the same path.

Lemma 8.3. *If p is a finite region path in $\mathcal{R}_N(\mathcal{G})$, the set of cumulated weights $\{\mathsf{wt}_\Sigma(\rho) \mid \rho$ play of $\mathcal{G}$ following $\mathsf{p}\}$ is an interval bounded by the minimum and the maximum values of the set $\{\mathsf{wt}_\Sigma(\overline{\rho}) \mid \overline{\rho}$ corner play of $\Gamma_N(\mathcal{G})$ following $\mathsf{p}\}$.*

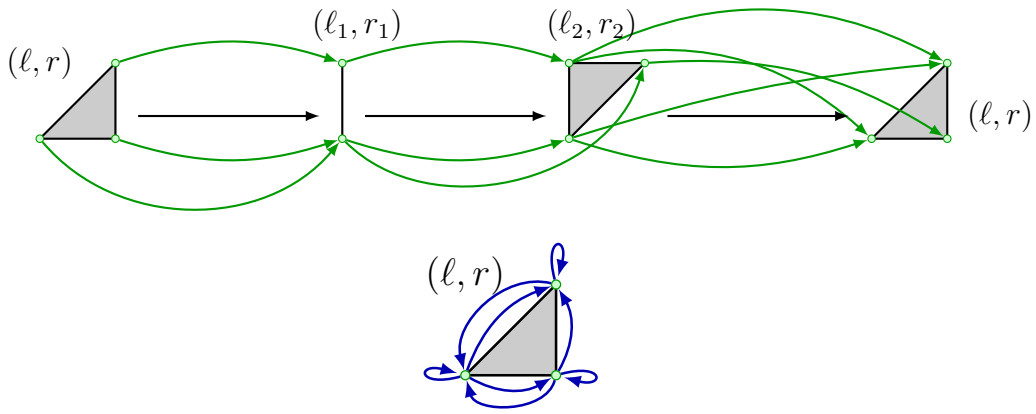


Figure 8.3.: A region cycle p in the region game (in black), its associated corner plays (in green), and its folded orbit graph (in blue). Note that there is no edge between the top right corner and the bottom right corner, as no corner play goes from the former to the latter.

Proof. The set $\{\mathsf{wt}_\Sigma(\rho) \mid \rho \text{ finite play following } \mathsf{p}\}$ is an interval as the image of a convex set by an affine function (see [BBBR07, Sec. 3.2] for an explanation).

The good properties of the corner-point abstraction allow us to conclude, since for every play ρ following p, one can find a corner play following p of smaller weight and one of larger weight [BBL08], and for every corner play ρ following p and every $\varepsilon > 0$, one can find a play following p whose weight is at most ε away from $\mathsf{wt}_\Sigma(\rho)$ by Lemma 8.2. □

An important property of the corner-point abstraction, derived from Proposition 2.1, is that corner plays cannot get stuck as long as they follow a region path:

Lemma 8.4. *Let p be a region path starting from (ℓ, r) and ending in (ℓ', r'). For all corners v of r, there exists a corner play following p that starts in (ℓ, r, v). For all corners v' of r' there exists a corner play following p that ends in (ℓ', r', v').*

Proof. Pick a valuation arbitrarily close to v. By Proposition 2.1, there exists a play that follows p starting from this valuation. As plays can be expressed as linear combinations of corner plays (see [Pur00] for details), there exists a corner play that starts in v, and similarly one that ends in v'. □

Useful theoretical tools stem from the corner-point abstraction. Notably, let us focus on a cycle of the region automaton. In order to study some properties of the corner plays following this cycle, we only need to consider the aggregation of all the behaviours following it. Inspired by the *folded orbit graphs* (FOG) introduced in [Pur00], we define the folded orbit graph $\mathsf{FOG}(\mathsf{p})$ of a region cycle $\mathsf{p} = (\ell_1, r = r_1) \xrightarrow{e_1} (\ell_2, r_2) \xrightarrow{e_2} \cdots \xrightarrow{e_n} (\ell_1, r)$ in $\mathcal{R}_N(\mathcal{G})$ as a graph whose vertices are the corners states of region r, and that contains an edge from corner v to corner v' if there exists a corner play $\overline{\rho}$ from (ℓ_1, r, v) to (ℓ_1, r, v') following p. We fix $\overline{\rho}$ arbitrarily and label the edge between v and v' in the FOG by this corner play: it is then denoted by $\mathsf{v} \xrightarrow{\overline{\rho}} \mathsf{v}'$. An example is depicted in Figure 8.3.

The folded orbit graph inherits interesting topological properties from the corner-point abstraction, notably, by Lemma 8.4, for all vertices v, there exists at least one outgoing edge $\mathsf{v} \xrightarrow{\overline{\rho}'} \mathsf{v}'$, and at least one incoming edge $\mathsf{v}'' \xrightarrow{\overline{\rho}''} \mathsf{v}$ in $\mathsf{FOG}(\mathsf{p})$.

8.1.2. Problems

As in weighted (untimed) games, we consider the *value problem*, mimicked from the one in $[\![\mathcal{G}]\!]$. Precisely, given a weighted timed game $\mathcal{G}$, a configuration (ℓ, ν) and a threshold $\alpha \in \mathbb{Z}_\infty$, we want to know whether $\mathsf{Val}_{\mathcal{G}}(\ell, \nu) \leqslant \alpha$. In the context of timed games, optimal strategies may not exist, even for finite values.[5] We generally focus on ε-optimal strategies, that guarantee the optimal value, up to a small error $\varepsilon \in \mathbb{R}_{>0}$. Moreover, when the value problem is undecidable, we also consider the *value approximation problem* that consists, given a precision $\varepsilon \in \mathbb{Q}_{>0}$, in computing an ε-approximation of $\mathsf{Val}_{\mathcal{G}}(\ell, \mathbf{0})$.

8.1.3. Related work

In the one-player case[6], computing the optimal value and an ε-optimal strategy for weighted timed automata is known to be PSPACE-complete [BBBR07]. In the two-player case, the value problem of WTGs (also called priced timed games in the literature) is undecidable with 3 clocks [BBR05, BJM15], or even 2 clocks in the presence of negative weights [BGNK+14] (for the existence problem, asking if a strategy of player Min can guarantee a given threshold). To obtain decidability, one possibility is to limit the number of clocks to 1: then, there is an exponential-time algorithm to compute the value as well as ε-optimal strategies in the presence of non-negative weights only [BBM06, Rut11, HIJM13], whereas the problem is only known to be PTIME-hard. A similar result can be lifted to arbitrary weights, under restrictions on the resets of the clock in cycles [BGH+15]. The other possibility to obtain a decidability result [BCFL04, ABM04] is to enforce a semantical property of divergence, originally called strictly non-Zeno cost: it asks that every play following a cycle in the region automaton has weight at least 1.

Other objectives, not directly related to optimal reachability, have been considered in [BCR14] for weighted timed games, like mean-payoff and parity objectives. In this work, the authors manage to solve these problems for the so-called class of δ-robust WTGs that they introduce.

[5] For example, a player may want to let time elapse as much as possible, but with delay $d < 1$ because of a strict guard.

[6] When all locations belong to the same player

9. Analysable classes of WTGs

In this chapter, we introduce several classes of weighted timed games, and study some properties of their region cycles. We focus on region cycles because the value problem is decidable when $\mathcal{R}(\mathcal{G})$ is acyclic [TMM02]. In contrast, region cycles authorise executions that accumulate weight in ways that are hard to analyse, classically called Zeno behaviours.

9.1. Main results

9.1.1. On the value problem

Let us start with the class of weighted timed games studied in [BCFL04], to our knowledge the greatest class of WTG where the value problem is known to be decidable.

Definition 9.1. A weighted timed game $\mathcal{G}$ with non-negative weights satisfies the *strictly non-Zeno cost* property when every finite play ρ in $\mathcal{G}$ following a cycle in the region automaton $\mathcal{R}(\mathcal{G})$ satisfies $\mathsf{wt}_\Sigma(\rho) \geqslant 1$.

The intuition behind this class is that the weight of any long enough execution in $\mathcal{G}$ will ultimately grow above any fixed bound, and diverge towards $+\infty$ for an infinite execution. Therefore, the value of $\mathcal{G}$ is equal to Val^i for some horizon i large enough, making the value problem decidable. It is shown in [BCFL04] that i can be bounded exponentially in the size of $\mathcal{G}$.

We introduce divergent weighted timed games, as an extension of divergent weighted games to the timed setting, that naturally generalises the strictly non-Zeno cost property to weights in $\mathbb{Z}$.

Definition 9.2. A weighted timed game $\mathcal{G}$ is *divergent* when every finite play ρ in $\mathcal{G}$ following a cycle in the region automaton $\mathcal{R}(\mathcal{G})$ satisfies $\mathsf{wt}_\Sigma(\rho) \notin (-1, 1)$.

Compared with the untimed notion of divergence, the cumulated weight is not only supposed to be different from 0, but also far from 0: otherwise, the original intuition on the ultimate growing of the weight of plays would not be fulfilled. If $\mathcal{G}$ has only non-negative weights on locations and edges, this definition matches with the strictly non-Zeno cost property of [BCFL04], we will therefore refer to their class as the class of divergent WTG with non-negative weights.

Remark. As in [BCFL04], we could replace $(-1, 1)$ by $(-\kappa, \kappa)$ to define a notion of κ-divergence. However, since weights and guard constraints in weighted timed games are integers, for $\kappa \in (0, 1)$, a weighted timed game $\mathcal{G}$ is κ-divergent if and only if it is divergent. This will be formally implied by Proposition 9.3 and Lemma 8.3.

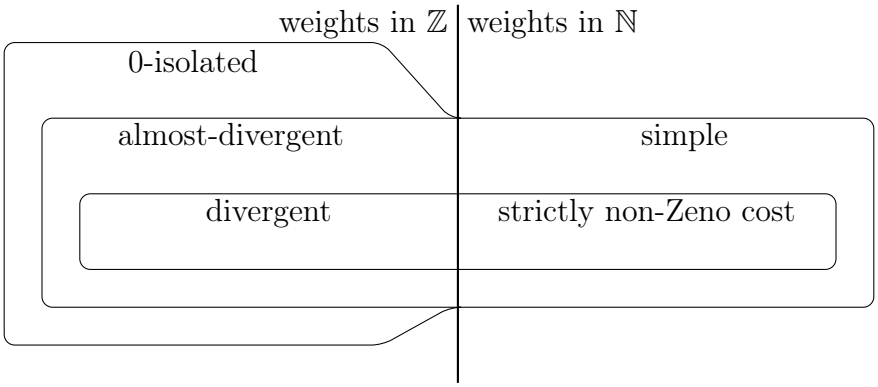


Figure 9.1.: Classes of weighted timed games, and their respective restrictions to non-negative weights.

We study this class in Chapter 10, where our contributions summarise as follows:

Theorem 9.1. *The value problem over divergent weighted timed games is decidable in* 3-EXPTIME*, and is* EXPTIME*-hard. Moreover, deciding if a given weighted timed game is divergent is a* PSPACE*-complete problem.*

9.1.2. On the value approximation problem

In [BJM15], the authors slightly extend the strictly non-Zeno cost property, to allow for cycles of weight exactly 0 while still preventing those of weight arbitrarily close to 0:

Definition 9.3. A weighted timed game $\mathcal{G}$ with non-negative weights is called *simple* when every finite play ρ in $\mathcal{G}$ following a cycle in the region automaton $\mathcal{R}(\mathcal{G})$ satisfies $\mathsf{wt}_\Sigma(\rho) \in \{0\} \cup [1, +\infty)$.

Unfortunately, it is shown in [BJM15] that the value problem is undecidable for simple WTGs. They propose a solution to the value approximation problem, as a procedure computing an approximation of the value of every configuration. The intuition is that cycles of weight exactly 0 are only possible when every (non-negative) weight encountered along the cycle equals 0, allowing one to define a subgame where every cyclic execution has weight 0. One can then analyse this subgame separately, by applying a semi-unfolding procedure on $\mathcal{R}(\mathcal{G})$, similar to the one presented in Chapter 7.4.

We now introduce a class of WTG that will extend the notion of simple WTG and allow negative weights:

Definition 9.4. A weighted timed game $\mathcal{G}$ is *0-isolated* when every finite play ρ in $\mathcal{G}$ following a cycle in the region automaton $\mathcal{R}(\mathcal{G})$ satisfies:

$$\mathsf{wt}_\Sigma(\rho) \in (-\infty, -1] \cup \{0\} \cup [1, +\infty)\,.$$

In other words, cyclic executions of weight exactly 0 are allowed, but not those close to 0. Clearly, every divergent WTG is 0-isolated, and, when weights are non-negative,

this class matches the simple WTGs of [BJM15], therefore inheriting their undecidability result.

We did not obtain positive results for the value approximation problem on this class of WTG. Instead, inspired by the almost-divergent class studied in the untimed setting, we restrict the 0-isolated class with a stability by decomposition requirement for cycles of weight 0.

If p is a region cycle in $\mathcal{R}(\mathcal{G})$, it is either simple or it can be decomposed into smaller region cycles p' and p'', such that (a rotation of) p equals $\mathsf{p}'\mathsf{p}''$.[1] Likewise, if ρ is a play in $\mathcal{G}$ following p, either p is simple, or ρ can be decomposed into smaller plays ρ' and ρ'' following respectively p' and p'', such that $\mathsf{wt}_\Sigma(\rho) = \mathsf{wt}_\Sigma(\rho') + \mathsf{wt}_\Sigma(\rho'')$.

Definition 9.5. A weighted timed game $\mathcal{G}$ is *almost-divergent* if every play ρ following a cycle p of $\mathcal{R}(\mathcal{G})$ satisfies either $\mathsf{wt}_\Sigma(\rho) \notin (-1, 1)$, or $\mathsf{wt}_\Sigma(\rho) = 0$ and for every decomposition of ρ into plays ρ' and ρ'' following smaller region cycles, $\mathsf{wt}_\Sigma(\rho') = \mathsf{wt}_\Sigma(\rho'') = 0$.[2]

By definition, the almost-divergent class of WTGs contains the divergent one, and is included in the 0-isolated class. When weights are non-negative, the stability by decomposition requirement for cycles of cumulated weight 0 always hold, as $\mathsf{wt}_\Sigma(\rho') + \mathsf{wt}_\Sigma(\rho'') = 0$ implies $\mathsf{wt}_\Sigma(\rho') = \mathsf{wt}_\Sigma(\rho'') = 0$. In this case, the almost-divergent and 0-isolated notions are thus equivalent, and the almost-divergent class matches with the simple WTGs of [BJM15]. We will therefore refer to simple WTGs as almost-divergent WTGs with non-negative weights. Figure 9.1 represents the hierarchy of the classes of WTG that we introduced.

Remark. Given a finite weighted game $\mathcal{G}$, we define its timed version as a weighted timed game, whose locations are the states of $\mathcal{G}$ equipped with weight 0, whose edges are the transitions of $\mathcal{G}$, enriched with guard $\top$ and reset $\mathcal{X}$, and of weight the weight of the transition in $\mathcal{G}$, with a single clock x. With this procedure, a divergent (resp. almost-divergent) weighted game becomes a divergent (resp. almost-divergent) weighted timed game. A weighted game that is not almost-divergent (for example, two loops of respective weight 1 and -1 on the same state) will become a WTG that is not almost-divergent, but that is 0-isolated: every state has weight 0, so the weights of (cyclic) executions are integers, and therefore not in $(-1, 0) \cup (0, 1)$. Thus, every class inclusion displayed in Figure 9.1 is strict.

We study almost-divergent games in Chapter 11. Our first result is the following extension of the approximation procedure for non-negative weights:

Theorem 9.2. *Given an almost-divergent WTG $\mathcal{G}$, a location ℓ and $\varepsilon \in \mathbb{Q}_{>0}$, we can compute an ε-approximation of $\mathsf{Val}_\mathcal{G}(\ell, \mathbf{0})$ in time triply-exponential in the size of $\mathcal{G}$ and polynomial in $1/\varepsilon$. Moreover, deciding if a WTG is almost-divergent is* PSPACE*-complete.*

[1] We refer to the cycle decomposition procedure described in Chapter 7.4 for finite transition systems. It is here applied on the region abstraction, seen as a finite transition system.

[2] Once again, we could replace $(-1, 1)$ by $(-\kappa, \kappa)$ with $0 < \kappa < 1$ to define an equivalent notion of κ-almost-divergence.

It heavily relies on the region abstraction, and requires one to construct $\mathcal{R}(\mathcal{G})$ entirely and compute its strongly connected components, before unfolding it partially in a tree-shaped structure. Our second result is a more symbolic approximation schema based on the value iteration algorithm only: the computations are not performed on the region abstraction, but instead use polyhedra that can cover several regions.

Theorem 9.3. *Let $\mathcal{G}$ be an almost-divergent WTG such that $\mathsf{Val}_{\mathcal{G}}(\ell, \nu) > -\infty$ for every configuration (ℓ, ν). Then the sequence $(\mathsf{Val}^k_{\mathcal{G}})_{k \geqslant 0}$ converges towards $\mathsf{Val}_{\mathcal{G}}$ and for every $\varepsilon \in \mathbb{Q}_{>0}$, we can compute an integer P such that $\mathsf{Val}^P_{\mathcal{G}}$ is an ε-approximation of $\mathsf{Val}_{\mathcal{G}}$ for all configurations.*

Note that we have to control for configurations (ℓ, ν) of value $-\infty$, where the non-increasing sequence $(\mathsf{Val}^k_{\mathcal{G}}(\ell, \nu))_{k \in \mathbb{N}}$ (that starts at $+\infty$) will converge towards $-\infty$, but has no hope of approximating it.

However, we will show that the configurations with value $-\infty$ can be computed pre-emptively:

Proposition 9.1. *In an almost-divergent weighted timed game $\mathcal{G}$, the value problem with threshold $-\infty$ is* EXPTIME*-complete.*

This contrasts with the general case, where this problem is undecidable:

Proposition 9.2. *Given a WTG $\mathcal{G}$ (not necessarily almost-divergent) and an initial location ℓ_0, the decision problem asking whether $\mathsf{Val}_{\mathcal{G}}(\ell_0, \mathbf{0}) = -\infty$ is undecidable.*

Proof. The proof goes via a reduction to the existence problem on turn-based WTG: given a WTG $\mathcal{G}$ (without final weight function), an integer threshold α and a starting location ℓ_0, does there exist a strategy for Min that can guarantee reaching the unique target location ℓ_t from ℓ_0 with weight $< \alpha$. In the non-negative setting, it is proved in [BBM06] that the problem is undecidable for the comparison $\leqslant \alpha$. In the negative setting, formal proofs are given for all comparison signs in [BGNK+14].

Consider $\mathcal{G}'$ the WTG built from $\mathcal{G}$ by adding a transition from ℓ_t to ℓ_0, without guards and resetting all the clocks, of discrete weight $-\alpha$. We add a new target location ℓ'_t, and add transitions of weight 0 from ℓ_t to ℓ'_t. Location ℓ_t and ℓ'_t belong to Min. Let us prove that $\mathsf{Val}_{\mathcal{G}'}(\ell_0, \mathbf{0}) = -\infty$ if and only if Min has a strategy to guarantee a weight $< \alpha$ in $\mathcal{G}$. Assume first $\mathsf{Val}_{\mathcal{G}'}(\ell_0, \mathbf{0}) = -\infty$. If $\mathsf{Val}_{\mathcal{G}}(\ell_0, \mathbf{0}) = -\infty$, we are done. Otherwise, Min must follow in $\mathcal{G}'$ the new transition from ℓ_t to ℓ_0 to enforce a cycle of negative value, and thus enforce a play from $(\ell_0, \mathbf{0})$ to ℓ_t with weight less than α. Therefore, there exists a strategy for Min in $\mathcal{G}$ that can guarantee a weight $< \alpha$. Reciprocally, if there exists a strategy for Min that can guarantee a weight $< \alpha$, then Min can force a negative cycle play and $\mathsf{Val}_{\mathcal{G}'}(\ell_0, \mathbf{0}) = -\infty$. □

9.2. Cycle-based analysis

In this section, we will study properties that region cycles must satisfy in divergent or almost-divergent WTGs. This will give us a better understanding of the modelling power these classes confer.

9.2.1. Cycles in a 0-isolated WTG

Let us start with properties that hold for all 0-isolated weighted timed games $\mathcal{G}$. Keeping the terminology of the untimed setting, a cycle p of $\mathcal{R}(\mathcal{G})$ is said to be a positive cycle (resp. a negative cycle, a 0-cycle) if every finite play ρ following p satisfies $\mathsf{wt}(\rho) \geqslant 1$ (resp. $\mathsf{wt}(\rho) \leqslant -1$, $\mathsf{wt}(\rho) = 0$).

We start by showing that, in a 0-isolated game, all cycles $\mathsf{p} = t_1 \cdots t_n$ of $\mathcal{R}(\mathcal{G})$ (with $t_1, \ldots, t_n$ edges of $\mathcal{R}(\mathcal{G})$) are either 0-cycles, positive cycles or negative cycles, and we can classify a cycle by looking only at one of the corner plays following it:

Lemma 9.1. *In a 0-isolated WTG, a cycle p is a positive cycle (resp. a negative cycle, a 0-cycle) if and only if there exists a corner play $\overline{\rho}$ following p with $\mathsf{wt}_\Sigma(\overline{\rho}) > 0$ (resp. $\mathsf{wt}_\Sigma(\overline{\rho}) < 0$, $\mathsf{wt}_\Sigma(\overline{\rho}) = 0$).*

Proof. If p is a positive cycle (resp. a negative cycle, a 0-cycle), every such corner play $\overline{\rho}$ will have weight above 0 (resp. under 0, equal to 0), by Lemma 8.3. Reciprocally, if such a corner play exists, all corner plays following p have weight above 0 (resp. under 0, equal to 0): otherwise the set $\{\mathsf{wt}_\Sigma(\rho) \mid \rho \text{ play following } \mathsf{p}\}$ would have non-empty intersection with the set $(-1, 0) \cup (0, 1)$ by Lemma 8.3, which would contradict that the game is 0-isolated. □

Corollary 9.1. *A WTG $\mathcal{G}$ is 0-isolated if and only if every region cycle in $\mathcal{G}$ is either positive, negative, or a 0-cycle.*

An important result is that the sign of cycles is stable by rotation. This is not trivial because plays following a cycle can start and end in different valuations, therefore changing the starting region state of the cycle could *a priori* change the plays that follow it and the sign of their weights.

Lemma 9.2. *Let p and p' be region paths of a 0-isolated WTG. If pp' is a positive cycle (resp. a negative cycle, a 0-cycle), then $\mathsf{p}'\mathsf{p}$ is a positive cycle (resp. a negative cycle, a 0-cycle).*

Proof. Since $\mathsf{p}_1 = \mathsf{pp}'$ is a cycle, $\mathsf{first}(\mathsf{p}) = \mathsf{last}(\mathsf{p}')$ and $\mathsf{first}(\mathsf{p}') = \mathsf{last}(\mathsf{p})$, so $\mathsf{p}_2 = \mathsf{p}'\mathsf{p}$ is a cycle as well. First, since there are finitely many corners, by constructing a long enough play following an iterate of $\mathsf{p}'\mathsf{p}$, we can obtain a corner play that starts and ends in the same corner. Formally, we define two sequences of region corners $(\mathsf{v}_i \in \mathsf{first}(\mathsf{p}))_i$ and $(\mathsf{v}'_i \in \mathsf{first}(\mathsf{p}'))_i$. We start by choosing any $\mathsf{v}_0 \in \mathsf{first}(\mathsf{p})$. Let v'_0 be a corner of $\mathsf{first}(\mathsf{p}')$ such that v'_0 is accessible from v_0 by following p with a corner play $\overline{\rho}_0$. For every $i > 0$, let v_i be a corner of $\mathsf{first}(\mathsf{p})$ such that v_i is accessible from v'_{i-1} by following p' with with a corner play $\overline{\rho}'_i$, and let v'_i be a corner of $\mathsf{first}(\mathsf{p}')$ such that v'_i is accessible from v_i by following p with a corner play $\overline{\rho}_i$. We stop the construction at the first index l such that there exists $k < l$ with $\mathsf{v}_l = \mathsf{v}_k$. Additionally, we let $\overline{\rho}_l = \overline{\rho}_k$. We know that this process never gets stuck—*i.e.* we can always find such corner plays iteratively—by Lemma 8.4, and it is bounded since $\mathsf{first}(\mathsf{p})$ has at most $|\mathcal{X}| + 1$ corners.

For every $0 \leqslant i \leqslant l$, let w_i be the weight of the corner play $\overline{\rho}_i$ from v_i to v'_i along p, and let w'_i be the weight of the corner play $\overline{\rho}'_i$ from v'_i to v_{i+1} along p'. The concatenation of

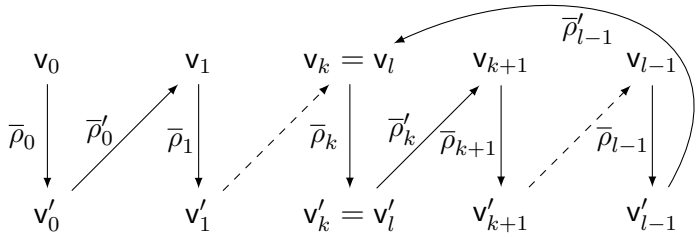


Figure 9.2.: Proof scheme of Lemma 9.2. The top labels are corners of $\mathsf{first}(\mathsf{p})$, the bottom ones are corners of $\mathsf{first}(\mathsf{p}')$, and edges represent corner plays.

the two plays has weight $w_i + w'_i > 0$ (resp. $w_i + w'_i < 0$, $w_i + w'_i = 0$), since it follows the positive cycle (resp. negative cycle, 0-cycle) p_1. For every $0 \leqslant i < l$, the concatenation of the corner play $\overline{\rho}'_i$ from v'_i to v_{i+1} with the corner play $\overline{\rho}_{i+1}$ from v_{i+1} to v'_{i+1} is a play from v'_i to v'_{i+1}, of weight $w'_i + w_{i+1}$, following p_2. Since p_2 is a cycle, and the game is 0-isolated, all possible values of $w'_i + w_{i+1}$ have the same sign by Lemma 9.1.

Finally, we can construct a corner play from v'_k to v'_l by concatenating the plays $\overline{\rho}'_k, \overline{\rho}_{k+1}, \dots, \overline{\rho}_{l-1}, \overline{\rho}'_{l-1}, \overline{\rho}_l$. We denote the weight of that play W, and

$$W = \sum_{i=k}^{l-1} (w'_i + w_{i+1}) = \sum_{i=k}^{l-1} (w_i + w'_i)$$

since $w_k = w_l$, and as $w_i + w'_i > 0$ (resp. $w_i + w'_i < 0$, $w_i + w'_i = 0$) holds for every i, we obtain $W > 0$ (resp. $W < 0$, $W = 0$).

This implies that the terms $w'_i + w_{i+1}$, of constant sign, are all above 0 (resp. under 0, equal to 0). As a consequence, the concatenation of $\overline{\rho}'_k$ and $\overline{\rho}_{k+1}$ is a corner play following p_2 of weight above 0 (resp. under 0, equal to 0). By Lemma 9.1, we conclude that p_2 must be a positive cycle (resp. a negative cycle, a 0-cycle). □

Therefore, region cycles in 0-isolated games are well-behaved: we can compose and rotate them while preserving their sign in the expected way. This will give us access to combinatorial proofs that extend the intuitions coming from the untimed setting.

9.2.2. SCC-based characterisations

As divergent WTG are 0-isolated, Lemma 9.1 lets us express divergence in terms of region cycles instead of cyclic executions:

Corollary 9.2. *A WTG $\mathcal{G}$ is divergent if and only if it is 0-isolated and contains no 0-cycles. In other words, if and only if every region cycle in $\mathcal{G}$ is either positive or negative.*

Similarly, we can express almost-divergence as a property of $\mathcal{R}(\mathcal{G})$ by applying Lemma 9.1:

Corollary 9.3. *A WTG $\mathcal{G}$ is almost-divergent if and only if it is 0-isolated and every 0-cycle p of $\mathcal{R}(\mathcal{G})$ satisfies the following property: for every decomposition of p into smaller region cycles p' and p'', p' and p'' are 0-cycles. In other words, if and only if every region cycle in $\mathcal{G}$ is either positive, negative, or a 0-cycle stable by decomposition.*

After studying the properties of region cycles, we now focus on strongly connected components (SCCs) of the region abstraction $\mathcal{R}(\mathcal{G})$. Following the notations of Chapter 7, an SCC S of $\mathcal{R}(\mathcal{G})$ is said to be positive (resp. negative) if every cycle in S is positive (resp. negative), *i.e.* if every play ρ following a region cycle in S satisfies $\mathsf{wt}_\Sigma(\rho) \geqslant 1$ (resp. $\mathsf{wt}_\Sigma(\rho) \leqslant -1$). We will transfer in the timed setting Proposition 7.2:

Proposition 9.3. *A weighted timed game $\mathcal{G}$ is divergent if and only if, each SCC of $\mathcal{R}(\mathcal{G})$ is either positive or negative.*

Likewise, an SCC S of $\mathcal{R}(\mathcal{G})$ is said to be non-negative (resp. non-positive) if every region cycle in S is either a positive cycle or a 0-cycle (resp. either a negative cycle or a 0-cycle), *i.e.* every play ρ following a region cycle in S satisfies either $\mathsf{wt}_\Sigma(\rho) \geqslant 1$ or $\mathsf{wt}_\Sigma(\rho) = 0$ (resp. either $\mathsf{wt}_\Sigma(\rho) \leqslant -1$ or $\mathsf{wt}_\Sigma(\rho) = 0$). We obtain:

Proposition 9.4. *A WTG $\mathcal{G}$ is almost-divergent if and only if each SCC of $\mathcal{R}(\mathcal{G})$ is either non-negative or non-positive.*

Remark. Note that if $\mathcal{G}$ is divergent it has no 0-cycle, and Proposition 9.4 implies that each SCC of $\mathcal{R}(\mathcal{G})$ is either positive or negative. Conversely, if each SCC of $\mathcal{R}(\mathcal{G})$ is either positive or negative, Proposition 9.4 implies that $\mathcal{G}$ is divergent. Therefore, Proposition 9.3 is a corollary of Proposition 9.4.

To prove the reciprocal implication of Proposition 9.4, we only need to show that non-negative (resp. non-positive) SCCs of $\mathcal{R}(\mathcal{G})$ are almost-divergent. By definition, such SCCs are 0-isolated, as they only contain executions ρ following region cycles such that $\mathsf{wt}_\Sigma(\rho) \in \{0\} \cup [1, +\infty)$ (resp. $\mathsf{wt}_\Sigma(\rho) \in (-\infty, -1] \cup \{0\}$). Then, if $\mathsf{wt}_\Sigma(\rho) = 0$ and ρ can be decomposed into smaller plays ρ' and ρ'' of non-negative (resp. non-positive) weight, it follows that $\mathsf{wt}_\Sigma(\rho') = \mathsf{wt}_\Sigma(\rho'') = 0$.

For the direct implication, the situation is more complex: we need to be more careful while composing cycles with each other, and weights in the timed game are no longer integers, forbidding the arithmetical reasoning we applied in the untimed setting. To help us, we rely on the folded orbit graphs of region cycles.

Suppose that $\mathcal{G}$ is almost-divergent, and consider two cycles p and p' in the same SCC of $\mathcal{R}(\mathcal{G})$. We need to show that they are both either non-positive or non-negative. Lemma 9.3 will first take care of the case where p and p' share a region state (ℓ, r).

Lemma 9.3. *If $\mathcal{G}$ is almost-divergent and two cycles p and p' of $\mathcal{R}(\mathcal{G})$ share a region state (ℓ, r), they are either both non-negative or both non-positive.*

Proof. Suppose by contradiction that p is negative and p' is positive. We assume that (ℓ, r) is the first region state of both p and p', possibly performing a rotation of the cycles if necessary (in particular this preserves their sign by Lemma 9.2). We construct

a graph $\mathsf{FOG}(\mathsf{p}, \mathsf{p}')$ as the union of $\mathsf{FOG}(\mathsf{p})$ and $\mathsf{FOG}(\mathsf{p}')$ (that share the same set of vertices), colouring in blue the edges of $\mathsf{FOG}(\mathsf{p})$ and in red the edges of $\mathsf{FOG}(\mathsf{p}')$. A path in $\mathsf{FOG}(\mathsf{p}, \mathsf{p}')$ is said blue (resp. red) when all of its edges are blue (resp. red).

We assume first that there exists in $\mathsf{FOG}(\mathsf{p}, \mathsf{p}')$ a blue cycle C and a red cycle C' with the same first vertex v (a corner of (ℓ, r)). Let k and k' be the respective lengths of C and C', so that C can be decomposed as $\mathsf{v} \xrightarrow{\overline{\rho}_1} \cdots \xrightarrow{\overline{\rho}_k} \mathsf{v}$ and C' as $\mathsf{v} \xrightarrow{\overline{\rho}'_1} \cdots \xrightarrow{\overline{\rho}'_{k'}} \mathsf{v}$, where $\overline{\rho}_i$ are corner plays following p and $\overline{\rho}'_i$ are corner plays following p'. Let $\overline{\rho}$ be the concatenation of $\overline{\rho}_1, \dots, \overline{\rho}_k$, and $\overline{\rho}'$ be the concatenation of $\overline{\rho}'_1, \dots, \overline{\rho}'_{k'}$. Recall that $w = |\mathsf{wt}_\Sigma(\overline{\rho})|$ and $w' = |\mathsf{wt}_\Sigma(\overline{\rho}')|$ are integers. Since p is negative, so is p^k, the concatenation of k copies of p (the weight of a play following it is a sum of weights all below -1). Therefore, $\overline{\rho}$, that follows p^k, has a weight $\mathsf{wt}_\Sigma(\overline{\rho}) \leqslant -1$ by Lemma 8.3. Similarly, $\mathsf{wt}_\Sigma(\overline{\rho}') \geqslant 1$. We consider the cycle C'' obtained by concatenating w' copies of C and w copies of C'. Similarly, we let $\overline{\rho}''$ be the play obtained by concatenating w' copies of $\overline{\rho}$ and w copies of $\overline{\rho}'$. Then, $\mathsf{wt}_\Sigma(\overline{\rho}'') = \mathsf{wt}_\Sigma(\overline{\rho})w' + \mathsf{wt}_\Sigma(\overline{\rho}')w = 0$, and therefore the region cycle p'' composed of w' copies p^k and w copies of $\mathsf{p}'^{k'}$ is a 0-cycle. This contradicts the almost-divergence of $\mathcal{G}$, since p'' can be decomposed into smaller cycles that are not 0-cycles.

We now return to the general case, where C and C' may not exist. Since $\mathsf{FOG}(\mathsf{p})$ and $\mathsf{FOG}(\mathsf{p}')$ are finite graphs with no deadlocks (every corner has an outgoing edge by Lemma 8.4), from every corner of $\mathsf{FOG}(\mathsf{p}, \mathsf{p}')$, we can reach a blue simple cycle, as well as a red simple cycle. Since there are only a finite number of simple cycles in $\mathsf{FOG}(\mathsf{p}, \mathsf{p}')$, there exists a blue cycle C and a red cycle C' that can reach each other in $\mathsf{FOG}(\mathsf{p}, \mathsf{p}')$. In $\mathsf{FOG}(\mathsf{p}, \mathsf{p}')$, we let P be a path from the first vertex of C to the first vertex of C', and P' be a path from the first vertex of C' to the first vertex of C. Consider the cycle C'' obtained by concatenating P and P'. As a cycle of $\mathsf{FOG}(\mathsf{p}, \mathsf{p}')$, we can map it to a cycle p'' of $\mathcal{R}(\mathcal{G})$ (alternating p and p' depending on the colours of the traversed edges), so that C'' is a cycle (of length 1) of $\mathsf{FOG}(\mathsf{p}'')$. As $\mathcal{G}$ is 0-isolated, p'' is either positive, negative or a 0-cycle. By the almost-divergence of $\mathcal{G}$, p'' cannot be a 0-cycle as it can be decomposed onto smaller cycles that are not 0-cycles. Suppose for instance that it is positive. Since (ℓ, r) is the first region state of both p and p'', we can construct the $\mathsf{FOG}(\mathsf{p}, \mathsf{p}'')$, in which C is a blue cycle and C'' is a red cycle, both sharing the same first vertex. We then conclude with the previous case. A similar reasoning with p' applies to the case that p'' is negative. Therefore, in all cases, we reached a contradiction. □

To finish the proof of the direct implication of Proposition 9.4, we suppose that the two cycles p and p', one positive and the other negative, in the same SCC of $\mathcal{R}(\mathcal{G})$, do not share any region states. By strong connectivity, in $\mathcal{R}(\mathcal{G})$, there exists a path p_1 from the first state of p to the first state of p', and a path p_2 from the first state of p' to the first state of p. Consider the cycle p'' of $\mathcal{R}(\mathcal{G})$ defined by $\mathsf{p}\mathsf{p}_1\mathsf{p}'\mathsf{p}_2$. By the almost-divergence of $\mathcal{G}$ and Corollary 9.1, p'' must be either positive, negative or a 0-cycle. Since it shares a state with both p and p', Lemma 9.3 allows us to prove a contradiction in both positive and negative cases, and therefore p'' must be a 0-cycle. This contradicts the hypothesis as one of the decompositions of p'' into smaller cycles produces p and $\mathsf{p}_1\mathsf{p}'\mathsf{p}_2$, with p a non-0-cycle. This concludes the proof of Proposition 9.4.

Remark. These characterisations of divergent or almost-divergent WTGs in term of SCCs provide an intuitive understanding of the modelling power these classes hold. For divergence, the model should have a global structure (the SCC decomposition) linking modules in an acyclic fashion. For each module, we have to choose between a positive dynamic, where weights always eventually increase, and a negative dynamic, where weights always eventually decrease. For almost-divergence, the modules may also have portions that are (eventually) neutral with regard to weight accumulation. In both classes, arbitrarily small weights should not be allowed to accumulate.

9.3. The membership problem

We study the *membership problem* for divergent (resp. almost-divergent) weighted timed games, *i.e.* the decision problem that asks if a given WTG is divergent (resp. almost-divergent). As mentioned in Theorems 9.1 and 9.2, we show that it is PSPACE-complete for both of these classes.

The proof is an extension of the NL bound of Lemmas 7.3 and 7.5 in the untimed setting, but this time we will guess region paths, hence the exponential blowup in complexity. In order to keep a compact representation of the weight of plays, we rely on the corner-point abstraction.

We show that, given a bound on the length of relevant region cycles, we can test in polynomial space the sign of corner plays:

Lemma 9.4. *Consider a weighted timed game $\mathcal{G}$, a region state (ℓ, r) of $\mathcal{R}(\mathcal{G})$, a bound $B \in \mathbb{N}$, and a comparison operator $\bowtie \in \{<, >, \leqslant, \geqslant, =, \neq\}$. Deciding if there exists a corner play $\overline{\rho}$ following a cycle p of $\mathcal{R}(\mathcal{G})$ starting from (ℓ, r), such that $|\mathsf{p}| \leqslant B$ and $\mathsf{wt}_\Sigma(\overline{\rho}) \bowtie 0$, is in* PSPACE.[3]

Proof. We guess a starting corner v of r for $\overline{\rho}$, and we guess on-the-fly the transitions of p and $\overline{\rho}$, *i.e.* a sequences of regions with one of their corners, keeping in memory the cumulated weight of $\overline{\rho}$ and the length $|\mathsf{p}|$. At every step, we check that $|\mathsf{p}| \leqslant B$ in space polynomial in $\log(B)$ and $\log(|\mathsf{p}|) \leqslant \log(|\mathcal{R}(\mathcal{G})|)$, with $\log(|\mathcal{R}(\mathcal{G})|)$ polynomial in $|\mathcal{G}|$.[4] Similarly, we can check that $\overline{\rho}$ is following p in polynomial space. At some point, we guess that the cycle is complete, and we check that the current region state equals (ℓ, r). Finally, we check that $\mathsf{wt}_\Sigma(\overline{\rho}) \bowtie 0$ in space polynomial in $\log(\mathsf{wt}_\Sigma(\overline{\rho}))$. Note that $\mathsf{wt}_\Sigma(\overline{\rho})$ is an integer bounded (in absolute value) by $B \times w_{\max}$, and can thus be stored in polynomial space.[5] This shows that the problem is in NPSPACE, and thus in PSPACE using Savitch's theorem [Sav70]. □

[3] *i.e.* it can be done using space polynomial in $|\mathcal{G}|$ and $\log(B)$.

[4] The global clock bound M is at most exponential in the size of $\mathcal{G}$, and $|\mathcal{R}(\mathcal{G})|$ is at most exponential in $|\mathcal{X}|$ but polynomial in M, therefore $|\mathcal{R}(\mathcal{G})|$ is at most exponential in $|\mathcal{G}|$.

[5] Compared with the untimed setting, we no longer care about the encoding of weights: if they are stored in binary, $w_{\max}$ is at most exponential in the size of $\mathcal{G}$, therefore $\log(w_{\max})$ is polynomial.

9.3.1. Deciding divergence

In order to solve the membership problem for divergent games, we will rely on Lemma 9.4, and on a characterisation of divergence based on region cycles of length bounded by the number of corners in the corner-point abstraction $\Gamma(\mathcal{G})$.

Lemma 9.5. *Let $\mathcal{G}$ be a weighted timed game. An SCC S of $\mathcal{R}(\mathcal{G})$ is positive (resp. negative) if and only if every region cycle in S, of length at most $|\Gamma(\mathcal{G})|$, is positive (resp. negative).*

Proof. The direct implication holds by definition. Reciprocally, let us assume that every cycle in S of length at most $|\Gamma(\mathcal{G})|$ is positive (resp. negative), and prove that every cycle p in S is positive (resp. negative), by induction on the length of p. If p has length above $|\Gamma(\mathcal{G})|$, every corner play $\overline{\rho}$ following p can be split as $\overline{\rho} = \overline{\rho}_1\overline{\rho}_2\overline{\rho}_3$, with $\overline{\rho}_2$ a corner play that starts and ends in the same corner. Then we can write $\mathsf{p} = \mathsf{p}_1\mathsf{p}_2\mathsf{p}_3$, with $\overline{\rho}_1$ (resp. $\overline{\rho}_2$, $\overline{\rho}_3$) following p_1 (resp. p_2, p_3). Observe that p_2 and $\mathsf{p}_1\mathsf{p}_3$ are region cycles of S, both positive (resp. negative) by induction. It follows that $\mathsf{wt}_\Sigma(\overline{\rho}_2) \geqslant 1$ (resp. $\mathsf{wt}_\Sigma(\overline{\rho}_2) \leqslant -1$), and $\mathsf{wt}_\Sigma(\overline{\rho}_1\overline{\rho}_3) \geqslant 1$ (resp. $\mathsf{wt}_\Sigma(\overline{\rho}_1\overline{\rho}_3) \leqslant -1$), as $\overline{\rho}_1\overline{\rho}_3$ is a valid corner play following $\mathsf{p}_1\mathsf{p}_3$. We can therefore conclude that $\mathsf{wt}_\Sigma(\overline{\rho}) \geqslant 1$ (resp. $\mathsf{wt}_\Sigma(\overline{\rho}) \leqslant -1$). This holds for all corner plays following p, and by Lemma 8.3 p is positive (resp. negative). □

Let us show how to decide if a game is *not divergent*. By Proposition 9.3 and Lemma 9.5, it suffices to search for an SCC of the region automaton containing a cycle such that there exists a corner play following it of non-negative weight, and a cycle such that there exists a corner play following it of non-positive weight, both of length bounded by $B = |\Gamma(\mathcal{G})| \leqslant |L| \times |\mathsf{Reg}(\mathcal{X}, M)| \times (|\mathcal{X}| + 1)$.

We can test this condition in NPSPACE: we guess a starting region for each cycle, use standard reachability analysis [AD94] to check that they are in the same SCC of $\mathcal{R}(\mathcal{G})$ (in PSPACE), and use Lemma 9.4 with comparison $\geqslant 0$ and $\leqslant 0$, respectively, to check each cycle's sign. Since the bound B is at most exponential in $|\mathcal{G}|$, this last step is also in PSPACE.

This shows that the membership problem for divergent weighted timed games is in coNPSPACE = coPSPACE = PSPACE by the theorems of Immerman-Szelepcsényi [Imm88, Sze88] and Savitch [Sav70].

Let us now show the PSPACE-hardness (indeed the coPSPACE, which is identical) by a reduction from the reachability problem in a timed automaton. As in the untimed setting, we consider a timed automaton with a starting location and a different target location without outgoing edges. We construct from it a weighted timed game by distributing all locations to Min, and equipping all edges with weight 1, and all locations with weight 0. We also add a loop with weight -1 on the target, and an edge from the target location to the initial location with weight 0, both with guard $\top$ and resetting all clocks. Then, the weighted timed game is not divergent if and only if the target can be reached from the initial location in the timed automaton.

9.3.2. Deciding almost-divergence

Let us now focus on the membership problem for almost-divergent WTGs. We start by stating a variation of Lemma 9.4, that reason on a pair of corner plays:

Lemma 9.6. *Consider a weighted timed game $\mathcal{G}$, a region state (ℓ, r) of $\mathcal{R}(\mathcal{G})$, a bound $B \in \mathbb{N}$, and comparison operators $\bowtie, \bowtie' \in \{<, >, \leqslant, \geqslant, =, \neq\}$. Deciding if there exists a cycle p of $\mathcal{R}(\mathcal{G})$ starting from (ℓ, r), and two corner plays $\overline{\rho}$ and $\overline{\rho}'$, both following p, such that $|\mathsf{p}| \leqslant B$, $\mathsf{wt}_\Sigma(\overline{\rho}) \bowtie 0$ and $\mathsf{wt}_\Sigma(\overline{\rho}') \bowtie' 0$, is in* PSPACE.

Proof. We follow the same non-deterministic procedure as Lemma 9.4, but this time we guess two corner plays on-the-fly instead of one. □

Then, we find a characterisation of almost-divergence based on region cycles of bounded length.

Lemma 9.7. *Let $\mathcal{G}$ be a weighted timed game. An SCC S of $\mathcal{R}(\mathcal{G})$ is non-negative (resp. non-positive) if and only if every region cycle in S, of length at most $|\Gamma(\mathcal{G})|^2$, is either a positive cycle or a 0-cycle (resp. either a negative cycle or a 0-cycle).*

Proof. The direct implication holds by definition. Reciprocally, suppose that every cycle in S of length at most $|\Gamma(\mathcal{G})|^2$ is either a positive cycle or a 0-cycle (resp. either a negative cycle or a 0-cycle). Let us prove that every cycle p in S is either a positive cycle or a 0-cycle (resp. either a negative cycle or a 0-cycle), by induction on the length of p. Consider a region cycle p with length above $|\Gamma(\mathcal{G})|^2$. Let us show that for all corner plays $\overline{\rho}$, $\overline{\rho}'$ following p, either $\mathsf{wt}_\Sigma(\overline{\rho}) = \mathsf{wt}_\Sigma(\overline{\rho}') = 0$, or both $\mathsf{wt}_\Sigma(\overline{\rho}) \geqslant 1$ and $\mathsf{wt}_\Sigma(\overline{\rho}') \geqslant 1$ (resp. both $\mathsf{wt}_\Sigma(\overline{\rho}) \leqslant -1$ and $\mathsf{wt}_\Sigma(\overline{\rho}') \leqslant -1$) hold. This will allow us to conclude by Lemma 8.3.

From a pair of corner plays $\overline{\rho}$ and $\overline{\rho}'$ following p, we can extract a sequence of pairs of corners states $((\ell_i, r_i, \mathsf{v}_i), (\ell_i, r_i, \mathsf{v}'_i))$, such that (ℓ_i, r_i) is the i-th region state of p, and v_i (resp. v'_i) is the i-th corner of $\overline{\rho}$ (resp. $\overline{\rho}'$). Since $|\mathsf{p}| > |\Gamma(\mathcal{G})|^2$, there must exist two indexes, j and k, such that $j < k$, $(\ell_j, r_j) = (\ell_k, r_k)$ and $(\mathsf{v}_j, \mathsf{v}'_j) = (\mathsf{v}_k, \mathsf{v}'_k)$. In other words, we can write $\mathsf{p} = \mathsf{p}_1\mathsf{p}_2\mathsf{p}_3$, with p_2 and $\mathsf{p}_1\mathsf{p}_3$ region cycles of S, and $\overline{\rho}$, $\overline{\rho}'$ can be split as $\overline{\rho} = \overline{\rho}_1\overline{\rho}_2\overline{\rho}_3$, $\overline{\rho}' = \overline{\rho}'_1\overline{\rho}'_2\overline{\rho}'_3$, with $\overline{\rho}_l$ (resp. $\overline{\rho}'_l$) following p_l (resp. p'_l) for $l \in \{1, 2, 3\}$, such that $\overline{\rho}_2$ and $\overline{\rho}'_2$ are corner cycles, *i.e.* $\mathsf{first}(\overline{\rho}_2) = \mathsf{last}(\overline{\rho}_2)$ and $\mathsf{first}(\overline{\rho}'_2) = \mathsf{last}(\overline{\rho}'_2)$. Then, by induction either $\mathsf{wt}_\Sigma(\overline{\rho}_2) = \mathsf{wt}_\Sigma(\overline{\rho}'_2) = 0$, or both $\mathsf{wt}_\Sigma(\overline{\rho}_2) \geqslant 1$ and $\mathsf{wt}_\Sigma(\overline{\rho}'_2) \geqslant 1$ (resp. both $\mathsf{wt}_\Sigma(\overline{\rho}_2) \leqslant -1$ and $\mathsf{wt}_\Sigma(\overline{\rho}'_2) \leqslant -1$) hold. The same property holds for $\overline{\rho}_1\overline{\rho}_3$ and $\overline{\rho}'_1\overline{\rho}'_3$, both valid corner plays following $\mathsf{p}_1\mathsf{p}_3$. It follows that either $\mathsf{wt}_\Sigma(\overline{\rho}) = \mathsf{wt}_\Sigma(\overline{\rho}') = 0$, or both $\mathsf{wt}_\Sigma(\overline{\rho}) \geqslant 1$ and $\mathsf{wt}_\Sigma(\overline{\rho}') \geqslant 1$ (resp. both $\mathsf{wt}_\Sigma(\overline{\rho}) \leqslant -1$ and $\mathsf{wt}_\Sigma(\overline{\rho}') \leqslant -1$) hold. □

Let us show how to decide if a game is *not almost-divergent*. By Proposition 9.4 and Lemma 9.7, we distinguish two cases for not being almost-divergent:

- There exists a region cycle, of length at most $B = |\Gamma(\mathcal{G})|^2$, and two corner plays $\overline{\rho}$ and $\overline{\rho}'$, both following p, such that $\mathsf{wt}_\Sigma(\overline{\rho}) = 0$ and $\mathsf{wt}_\Sigma(\overline{\rho}') \neq 0$.

- An SCC of the region automaton contains a cycle such that there exists a corner play following it of negative weight, and a cycle such that there exists a corner play following it of positive weight, both of length bounded by $B = |\Gamma(\mathcal{G})|^2$.

Once again, we can test both conditions in NPSPACE, by guessing the starting regions of these cycles and using respectively Lemmas 9.6 and 9.4.

This shows that the membership problem for divergent weighted timed games is in coNPSPACE = coPSPACE = PSPACE [Imm88, Sze88, Sav70].

Let us now show the PSPACE-hardness (indeed the coPSPACE, which is identical) by a reduction from the reachability problem in a timed automaton, similar to the one we used for the PSPACE-hardness of deciding divergence. We consider a timed automaton with a starting location and a different target location without outgoing edges. We construct from it a weighted timed game by distributing all locations to Min, and equipping all edges with weight 0, and all locations with weight 0. We also add a loop with weight 1 on the initial location, one with weight -1 on the target location, and an edge from the target location to the initial location with weight 0, all three with guard $\top$ and resetting all clocks. Then, the weighted timed game is not almost-divergent if and only if the target can be reached from the initial location in the timed automaton.

10. Computing values

In this chapter, we focus on classes of WTGs where the value problem is decidable. We will recall how one can compute values with bounded horizon $i \in \mathbb{N}$, by lifting the value iteration algorithm of finite weighted games to weighted timed games, hereby solving the value problem on acyclic WTGs. This has previously been done by [ABM04]. Instead of relying on the result of [ABM04] as is, we will give a detailed explanation of their techniques, with mostly independent proofs. The motivation for doing so are as follows:

- On the one hand, our setting is more general, in the sense that we allow for negative weights and for final weights, where they do not do so explicitly.
- On the other hand, their result is stated for concurrent games, a generalisation of the turn-based games we consider. This leads to simplifications in the proofs, and lowers some parts of the complexity analysis.
- We will need, in Chapter 11, to bound the partial derivatives of the functions we compute. This cannot be deduced from their result directly.
- We present their techniques in a new, more symbolic light, by performing computations on the entire state-space at once instead of region by region.
- Even by following their explanations closely, we are not able to replicate their complexity analysis, and detail possible reasons for this. We will therefore rely on a doubly-exponential upper bound instead of the exponential one claimed in [ABM04].

Then, we focus on divergent WTGs, and give a 3-EXPTIME decision procedure for computing values.

10.1. Symbolic value iteration

If $\boldsymbol{V}$ represents a value function—*i.e.* a mapping from configurations of $L \times \mathbb{R}_{\geqslant 0}^{\mathcal{X}}$ to a value in $\mathbb{R}_\infty$—we denote by $\boldsymbol{V}_\ell$ the mapping $\nu \mapsto \boldsymbol{V}(\ell, \nu)$. One step of the game is summarised in the following operator $\mathcal{F}$ mapping each value function $\boldsymbol{V}$ to a value function $\boldsymbol{V}' = \mathcal{F}(\boldsymbol{V})$ defined by $\boldsymbol{V}'_\ell(\nu) = \mathsf{wt}_{\mathsf{t}}(\ell, \nu)$ if $\ell \in L_{\mathsf{t}}$, and otherwise

$$\boldsymbol{V}'_\ell(\nu) = \begin{cases} \sup_{(\ell,\nu) \xrightarrow{d,e} (\ell',\nu')} \left[d \cdot \mathsf{wt}(\ell) + \mathsf{wt}(e) + \boldsymbol{V}_{\ell'}(\nu') \right] & \text{if } \ell \in L_{\mathsf{Max}} \\ \inf_{(\ell,\nu) \xrightarrow{d,e} (\ell',\nu')} \left[d \cdot \mathsf{wt}(\ell) + \mathsf{wt}(e) + \boldsymbol{V}_{\ell'}(\nu') \right] & \text{if } \ell \in L_{\mathsf{Min}} \end{cases} \tag{10.1}$$

where $(\ell, \nu) \xrightarrow{d,e} (\ell', \nu')$ ranges over valid transitions in $\mathcal{G}$. Then, starting from $\boldsymbol{V}^0$ mapping every configuration (ℓ, ν) to $+\infty$, except for the targets mapped to $\mathsf{wt}_{\mathsf{t}}(\ell, \nu)$, we let $\boldsymbol{V}^i = \mathcal{F}(\boldsymbol{V}^{i-1})$ for all $i > 0$. The value function $\boldsymbol{V}^i$ contains the value $\mathsf{Val}^i_{\mathcal{G}}$, which is what Min can guarantee when forced to reach the target in at most i steps [BCFL04, ABM04].

The rest of this section presents a symbolic algorithm computing $\mathsf{Val}^i_{\mathcal{G}}$ in time doubly-exponential in i and the size of $\mathcal{G}$. Intuitively, we observe that the mappings $\boldsymbol{V}^0_\ell$ are piecewise affine for all ℓ, and show that $\mathcal{F}$ preserves piecewise affinity, such that all iterates $\boldsymbol{V}^i_\ell$ can be computed using piecewise affine functions. In order to bound the size of $\boldsymbol{V}^i_\ell$ (in particular, its number of pieces), we need to get into the details of how the mappings $\boldsymbol{V}_\ell$ are encoded.

This result contrasts with [ABM04], where the value problem with bounded horizon is claimed to be in EXPTIME. This is discussed in Section 10.1.4.

10.1.1. Value functions as nested partitions

We now present the class of piecewise affine value functions, and a way to efficiently encode them, as developed in [ABM04]. In this section, n will denote the number of clocks, such that $\mathcal{X} = \{\mathsf{x}_1, \ldots, \mathsf{x}_n\}$, and $\ell \in L$ will be a location of $\mathcal{G}$. An *affine value function* is a mapping $\boldsymbol{V}_\ell : \mathbb{R}^{\mathcal{X}}_{\geqslant 0, <M} \to \mathbb{R}_\infty$ such that for all $\nu \in \mathbb{R}^{\mathcal{X}}_{\geqslant 0, <M}$,

$$\boldsymbol{V}_\ell(\nu) = \sum_{i=1}^{n} a_i \cdot \nu(\mathsf{x}_i) + b\,,$$

with partial derivatives $a_i \in \mathbb{Q}$ for $1 \leqslant i \leqslant n$, and additive constant $b \in \mathbb{Q}$. In this case, we say that $\boldsymbol{V}_\ell$ is defined by the equation

$$\mathsf{y} = \sum_{i=1}^{n} a_i \mathsf{x}_i + b\,,$$

where the variable $\mathsf{y} \notin \mathcal{X}$ refers to $\boldsymbol{V}_\ell(\mathsf{x}_1, \ldots, \mathsf{x}_n)$. We also consider infinite mappings $\nu \mapsto +\infty$ and $\nu \mapsto -\infty$ to be affine value functions, defined by $\mathsf{y} = \sum_{i=1}^{n} 0 \cdot \mathsf{x}_i + (+\infty)$ and $\mathsf{y} = \sum_{i=1}^{n} 0 \cdot \mathsf{x}_i + (-\infty)$, respectively.

Intuitively, we define a piecewise affine value function as a partition of $\mathbb{R}^{\mathcal{X}}_{\geqslant 0, <M}$ into finitely many polyhedra, called cells, each equipped by an affine value function. Formally, an affine inequality is an equation I of the form

$$\sum_{i=1}^{n} a_i \mathsf{x}_i + b \prec 0\,,$$

where $b \in \mathbb{Q}$ is the additive constant of I, $\prec \in \{<, \leqslant\}$ is its comparison operator, and for every $1 \leqslant i \leqslant n$, $a_i \in \mathbb{Q}$ is the i-th partial derivative of I. Similarly, an affine equality is

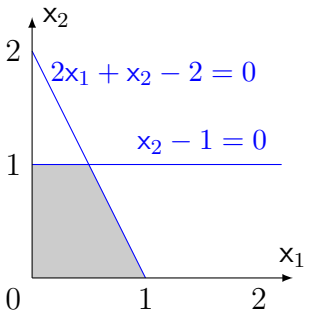


Figure 10.1.: The cell $2\mathsf{x}_1 + \mathsf{x}_2 - 2 < 0 \wedge \mathsf{x}_2 - 1 < 0$ in gray, and its borders in blue.

an equation E of the form

$$\sum_{i=1}^{n} a_i \mathsf{x}_i + b = 0\,.$$

We say that $\nu \in \mathbb{R}^{\mathcal{X}}_{\geqslant 0}$ satisfies I (resp. E), and write $\nu \models I$ (resp. $\nu \models E$), if $\sum_{i=1}^{n} a_i \cdot \nu(\mathsf{x}_i) + b \prec 0$ (resp. $= 0$) holds. In this case, $[\![I]\!]$ (resp. $[\![E]\!]$) refers to the set of valuations that satisfy I (resp. E). Equalities (resp. inequalities) are equivalent when they are satisfied by the same valuations. In particular, multiplying the additive constant b and all partial derivatives a_i by the same factor $N \in \mathbb{N}_{>0}$ gives an equivalent equality (resp. inequality), and we will therefore assume that they are always integers.

Definition 10.1. A *cell* is a set $c \subseteq \mathbb{R}^{\mathcal{X}}_{\geqslant 0}$, defined by a conjunction of affine inequalities $I_1 \wedge \cdots \wedge I_m$, such that $\nu \in c$ if and only if for all $1 \leqslant i \leqslant m$, $\nu \models I_i$. We write $c = [\![I_1 \wedge \cdots \wedge I_m]\!]$ in this case.

Cells are convex polyhedra, and the intersection of finitely many cells is a cell. From every affine inequality I we can extract an affine equality $E(I)$, of identical partial derivatives and additive constant. Then, we call *borders* of a cell $c = [\![I_1 \wedge \cdots \wedge I_m]\!]$ the affine equalities $E(I_1), \ldots E(I_m)$. The closure $\overline{c}$ of a cell c is obtained by replacing every comparison operator $<$ by $\leqslant$ in its affine inequalities. Note that regions and zones are particular cases of cells, where borders are of the form $\mathsf{x} + b = 0$ or $\mathsf{x} - \mathsf{x}' + b = 0$.

Let E be an affine equality of equation $\sum_{i=1}^{n} a_i \mathsf{x}_i + b = 0$. We say that $\mathbb{R}^{\mathcal{X}}_{\geqslant 0}$ is partitioned by E into three cells:

- $c_<$, defined by $\sum_{i=1}^{n} a_i \mathsf{x}_i + b < 0$;
- $c_>$, defined by $\sum_{i=1}^{n} a_i \mathsf{x}_i + b > 0$, *i.e.* $\sum_{i=1}^{n} -a_i \mathsf{x}_i - b < 0$;
- $c_=$, defined by $\sum_{i=1}^{n} a_i \mathsf{x}_i + b = 0$, *i.e.* $\sum_{i=1}^{n} a_i \mathsf{x}_i + b \leqslant 0 \wedge \sum_{i=1}^{n} -a_i \mathsf{x}_i - b \leqslant 0$.

Then, given a set $\mathcal{E} = \{E_1, \ldots, E_m\}$ of affine equalities, we denote $c^j_<$, $c^j_>$ and $c^j_=$ the three cells obtained from $E_j \in \mathcal{E}$. For every mapping $\phi : \mathcal{E} \to \{<, >, =\}$, we define c_ϕ as the cell $c^1_{\phi(E_1)} \cap \cdots \cap c^m_{\phi(E_m)}$. Every valuation of $\mathbb{R}^{\mathcal{X}}_{\geqslant 0}$ belongs to some c_ϕ, and if $\phi \neq \phi'$ then $c_\phi \cap c_{\phi'} = \emptyset$, such that the set of mappings $\{<, >, =\}^{\mathcal{E}}$ provides a partition of $\mathbb{R}^{\mathcal{X}}_{\geqslant 0}$

into 3^m cells. We say that $\mathbb{R}^{\mathcal{X}}_{\geqslant 0}$ is partitioned by $\mathcal{E}$ into $m' \in \mathbb{N}$ cells if m' of those 3^m cells are non-empty. In fact, m' is bounded by $\mathcal{O}(m^n)$ (see *e.g.* [Mat02]), and we denote $\mathsf{Splits}(m, n)$ this bound (polynomial in m and exponential in n) on the number of cells in the partition. Similarly, a cell $c \subseteq \mathbb{R}^{\mathcal{X}}_{\geqslant 0}$ is partitioned by $\mathcal{E}$ into at most $\mathsf{Splits}(m, n)$ sub-cells that have non-empty intersection with c. In particular, under the bounded clocks assumption we will partition $\mathbb{R}^{\mathcal{X}}_{\geqslant 0,<M}$ instead of $\mathbb{R}^{\mathcal{X}}_{\geqslant 0}$.

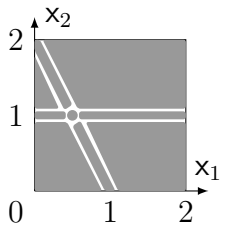


Figure 10.2.: A partition of $\mathbb{R}^{\mathcal{X}}_{\geqslant 0,<2}$ according to two affine equalities.

Example 10.1. The $\mathsf{Splits}(2, 2) = 9$ cells that partition $\mathbb{R}^{\mathcal{X}}_{\geqslant 0,<2}$ according to $\mathcal{E} = \{2\mathsf{x}_1 + \mathsf{x}_2 - 2 = 0, \mathsf{x}_2 - 1 = 0\}$ are represented in Figure 10.2.

Definition 10.2. A *nested partition* is a tree P, where nodes are labelled by a cell c and a set $\mathcal{E}$ of affine equalities, such that the children of c in P are labelled by the cells that partition c according to $\mathcal{E}$.

The cell c_P at the root of a nested partition P is called the domain of P. Leaves of P are equipped by empty sets of equalities, and their cells are called *base cells*. The domain c_P is partitioned by its base cells, and we denote $[\nu]_P$ the base cell that contains valuation $\nu \in c_P$.

Example 10.2. Figure 10.3 represents a nested partition P over domain $c_P = \mathbb{R}^{\mathcal{X}}_{\geqslant 0,<2}$. The root c_P is partitioned as in Figure 10.2, and has 9 children nodes. They are all base

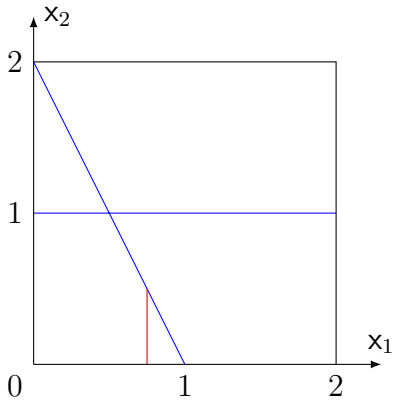


Figure 10.3.: A nested partition of $\mathbb{R}^{\mathcal{X}}_{\geqslant 0,<2}$.

cells, except the cell of Figure 10.1, that is further partitioned by the affine equality $4\mathsf{x}_1 - 3 = 0$ into three base cells.

Definition 10.3. A *partition value function* F defined over a nested partition P is a mapping from the base cells of P to affine value functions. It encodes a mapping from c_P to $\mathbb{R}_\infty$, denoted $[\![F]\!]$: if $\nu \in c_P$ and $F([\nu]_P)$ is defined by $\mathsf{y} = \sum_{i=1}^n a_i\mathsf{x}_i + b$, then $[\![F]\!](\nu)$, denoted $[\![F]\!]_\nu$, equals $\sum_{i=1}^n a_i \cdot \nu(\mathsf{x}_i) + b$.

A partition value function F of domain c_P is *continuous* if for all $\nu \in c_P$, for every base cell c_b such that $\nu \in \overline{c_b}$, if $F(c_b)$ is defined by $\mathsf{y} = \sum_{i=1}^n a_i\mathsf{x}_i + b$ then $[\![F]\!]_\nu = \sum_{i=1}^n a_i \cdot \nu(\mathsf{x}_i) + b$. In other words, the affine equations provided by F to neighbouring cells should match on the borders that separate them.

Finally, a *piecewise affine value function* $\boldsymbol{V}_\ell : \mathbb{R}^{\mathcal{X}}_{\geqslant 0,<M} \to \mathbb{R}_\infty$ is encoded as a pair (P, F) where P is a nested partition of domain $\mathbb{R}^{\mathcal{X}}_{\geqslant 0,<M}$, and where F is a partition value function defined over P, such that $[\![F]\!] = \boldsymbol{V}_\ell$.

A piecewise affine value function (P, F) is said *continuous on regions* if for every region $r \in \mathsf{Reg}(\mathcal{X}, M)$, the restriction of F to domain r is continuous. There could be discontinuities in F, but only at borders separating different regions. In particular, if a partition value function is continuous over regions, and $[\![F]\!]_\nu = +\infty$ (resp. $-\infty$) for some ν, then for all ν' in the same region as ν, $[\![F]\!]_{\nu'} = [\![F]\!]_\nu$.

Remark. In [ABM04], the domain of nested partitions is always a single region, and one value function is associated to each region. We define value functions over $\mathbb{R}^{\mathcal{X}}_{\geqslant 0,<M}$ instead, in order to obtain a symbolic algorithm. This induces slight differences in the way value functions are defined, because their mappings are continuous everywhere while ours can have discontinuities at borders between regions. In effect, they define their partitions with overlaps over borders, such that $\mathbb{R}^{\mathcal{X}}_{\geqslant 0}$ is partitioned by an affine equality into two cells, $c_\leqslant$ and $c_\geqslant$, instead of the three $c_<$, $c_>$ and $c_=$. This changes the number of cells $\mathsf{Splits}(m, n)$, but not asymptotically.

Cells are bounded, convex polyhedra. As such, elementary operations over cells (emptiness, intersection and inclusion tests) can be seen as instances of linear programming, and can thus be performed in polynomial time.

10.1.2. Operations over value functions

Let $\boldsymbol{V}$ be a piecewise affine value function, encoded as a pair (P_ℓ, F_ℓ) for each location $\ell \in L$, such that $[\![F_\ell]\!] = \boldsymbol{V}_\ell$. Our goal is to compute the value function $\mathcal{F}(\boldsymbol{V})$, as a pair (P'_ℓ, F'_ℓ) for each $\ell \in L$. In order to express complexity results for this operation (and its iteration that computes $\mathcal{F}^i(\boldsymbol{V})$), we need to bound the size of the pairs (P'_ℓ, F'_ℓ) with respect to the size of the pairs (P_ℓ, F_ℓ).

This requires a precise analysis, where one keeps track of the number of linear equalities in the nodes of the nested partition. In a nested partition P, the depth of a node is the distance to the root of P, and the depth of P is the greatest such depth. We say that P is a *k-nested partition* if P has depth at most $k - 1 \geqslant 0$, and nodes are on level i if their depth is $i - 1$ (in other words, the root is at level 1 and leaves at level depth plus one,

and a singleton partition where the root is a base cell is 1-nested). A k-nested partition P is said to be of *complexity* at most $\langle m \rangle$ if for every node of P labelled by a set $\mathcal{E}$ of affine equalities, it holds that $|\mathcal{E}| \leqslant m$. In this case, the number of base cells in P is bounded by $\mathsf{Splits}(m,n)^k$ (every node gets partitioned into at most $\mathsf{Splits}(m,n)$ sub-cells), and the number of nodes in P is bounded by $k \cdot \mathsf{Splits}(m,n)^k$. Let $\beta_P \in \mathbb{N}$ denote the greatest constant (partial derivatives and additive constant) in the affine equalities and inequalities of P, in absolute value, and let m_0 denote the number of inequalities in the encoding of the domain c_P.[1] Then, P can be stored in space

$$|P| \leqslant k \cdot \mathsf{Splits}(m,n)^k \cdot (m_0 + mk)(n+1)(\log(\beta_P)+1)\,,$$

as every node in P is labelled by a cell with at most $m_0 + m(k-1)$ borders and at most m affine equalities, and each affine expression is stored in space $(n+1)\lceil \log(\beta_P) \rceil \leqslant (n+1)(\log(\beta_P)+1)$.

For a partition value function F defined over P, we only need to additionally monitor the constants in affine value functions. Recall the affine equations in F are of the form $\mathsf{y} = \sum_{i=1}^{n} a_i \mathsf{x}_i + b$ with a_i and b in $\mathbb{Q}$. In order to monitor their size, we instead write $a_{\mathsf{y}} \mathsf{y} = \sum_{i=1}^{n} a_i \mathsf{x}_i + b$, with all a_i and b integers of $\mathbb{Z}$, and $a_{\mathsf{y}} \in \mathbb{N}_{>0}$, and ask that some $\beta_F \in \mathbb{N}$ bounds all of these constants in absolute value. Then, a value function F defined over a k-nested partition P of complexity at most $\langle m \rangle$ is stored in space

$$|F| \leqslant \mathsf{Splits}(m,n)^k \cdot (n+2)(\log(\beta_F)+1)\,.$$

We summarize these observations in the following lemma:

Lemma 10.1. *Let (P,F) encode a piecewise affine value function, such that (P,F) uses space $|(P,F)|$. Then, there exists k, m, β_P and β_F such that P is a k-nested partition of complexity at most $\langle m \rangle$, with constants bounded by β_P in P and β_F in F, with k, m at most linear in $|(P,F)|$, and β_P, β_F at most exponential in $|(P,F)|$.*

Reciprocally, let (P,F) be a k-nested partition of complexity at most $\langle m \rangle$, with constants bounded by β_P in P and β_F in F. It requires space at most logarithmic in β_P, β_F, polynomial in m, and exponential in k, n.

We will now introduce useful operations over nested partitions, and explain how they affect the depth, complexity, and constants.

If P_1 and P_2 are nested partitions over the same domain c_P, let $P_1 \oplus P_2$ denote the coarsest nested partition that refines both P_1 and P_2: each base cell c_b of $P_1 \oplus P_2$ corresponds to an intersection $c_1 \cap c_2$, with c_1 a base cell of P_1 and c_2 a base cell of P_2. It is obtained by exploring P_1 and P_2 in a top-down fashion, while creating corresponding nodes in $P_1 \oplus P_2$, starting from both root nodes. Let the current node in P_1 be n_1, labelled by c_1 and $\mathcal{E}_1$. Let the current node in P_2 be n_2, labelled by c_2 and $\mathcal{E}_2$. If $c_1 \cap c_2 \neq \emptyset$, we create a node in $P_1 \oplus P_2$ labelled by $c_1 \cap c_2$ and $\mathcal{E}_1 \cup \mathcal{E}_2$. The children of this node will be created inductively, by considering all pairs (n_1', n_2') such that n_1' is a child of n_1 in P_1 and n_2' is a child of n_2 in P_2, and continuing the top-down exploration at n_1' and

[1] If $c_P = \mathbb{R}^{\mathcal{X}}_{\geqslant 0, <M}$, then $m_0 = n$.

n_2'. If $c_1 \cap c_2 = \emptyset$, we do not create a node in $P_1 \oplus P_2$. Note that if P_1 and P_2 are both k-nested partitions of complexity at most $\langle m \rangle$ with constants bounded by β_P, $P_1 \oplus P_2$ is a k-nested partition of complexity at most $\langle 2m \rangle$ with constants bounded by β_P.

The minimum (resp. maximum) of a finite set of piecewise affine value functions can be computed with nested partitions.

Lemma 10.2 (Thm. 1, [ABM04]). *Let (P_i, F_i), $i = 1, \dots, q$ be q piecewise affine value functions, defined over the same domain c_P, where each P_i is a k-nested partition of complexity at most $\langle m \rangle$, with constants bounded by β_P in P and β_F in F. Then there exists a piecewise affine value function (P', F') of domain c_P, where P' is an atomic $(k+1)$-nested partition of complexity at most $\langle \max(qm, q^2) \rangle$, with constants bounded by $2\beta_P^2$ in P' and β_F in F', such that $[\![F']\!] = \min_{i=1,\dots,q} [\![F_i]\!]$ (resp. $[\![F']\!] = \max_{i=1,\dots,q} [\![F_i]\!]$).*

Proof. Let P' be $P_1 \oplus \cdots \oplus P_q$. Let c denote a base cell in P', corresponding to an intersection $c_1 \cap \cdots \cap c_q$ of base cells of $P_1, \dots, P_q$ respectively. Consider the affine value functions $F_1(c_1), \dots, F_q(c_q)$. Each of these is defined by an equation of the form $a_{\mathsf{y}}\mathsf{y} = \sum_{i=1}^n a_i \mathsf{x}_i + b$. We can see them as linear equalities over variables $\mathcal{X} \uplus \{\mathsf{y}\}$, denoted $E_1, \dots, E_q$, or equivalently as sets of valuations in $\mathbb{R}_{\geqslant 0}^{\mathcal{X} \uplus \{\mathsf{y}\}}$, denoted $[\![E_1]\!], \dots, [\![E_q]\!]$. If E, E' are such equalities, of equations $a_{\mathsf{y}}\mathsf{y} = \sum_{i=1}^n a_i \mathsf{x}_i + b$ and $a'_{\mathsf{y}}\mathsf{y} = \sum_{i=1}^n a'_i \mathsf{x}_i + b'$, the intersection $[\![E]\!] \cap [\![E']\!]$ is either empty or, by elimination of y, it satisfies the equation

$$\sum_{i=1}^{n} (a_{\mathsf{y}} a'_i - a'_{\mathsf{y}} a_i)\mathsf{x}_i + (a_{\mathsf{y}} b' - a'_{\mathsf{y}} b) = 0 \,.$$

This describes an affine equality over $\mathcal{X}$, that we denote $E \cap_y E'$. Now, let us partition c by the set of all such intersections

$$\mathcal{E} = \{E_i \cap_y E_j \mid i, j \in [1, q] \wedge [\![E_i]\!] \cap [\![E_j]\!] \neq \emptyset\} \,.$$

On every sub-cell c' in this partition, there exists $j \in [1, n]$ such that for every $\nu \in c'$, $[\![F_j]\!]_\nu = \min_{i=1,\dots,q} [\![F_i]\!]_\nu$. Therefore, we define F' on c' as equal to $F_j(c_j)$. The k-nested partition $P_1 \oplus \cdots \oplus P_q$ has complexity $\langle qm \rangle$, and we partitioned its base cells by at most $\langle q^2 \rangle$ intersections $E \cap_y E'$, resulting in a $(k+1)$-nested partition of complexity $\langle \max(qm, q^2) \rangle$. □

In order to implement the computation of $\mathcal{F}(\boldsymbol{V})$, we define intermediate operations over value functions. Let $\ell \in L$ be a location, of value function $\boldsymbol{V}_\ell$. For all $\nu \in \mathbb{R}^{\mathcal{X}}_{\geqslant 0, <M}$, let $d_\nu \in \mathbb{R}_{\geqslant 0}$ denote the greatest valid delay from ν, such that $d_\nu = \sup\{d \mid \nu + d \in \mathbb{R}^{\mathcal{X}}_{\geqslant 0, <M}\}$.[2] Consider the following operations:

- If $\mathcal{Y} \subseteq \mathcal{X}$ is a set of clocks, let $\mathsf{Unreset}_{\mathcal{Y}}(\boldsymbol{V}_\ell) : \mathbb{R}^{\mathcal{X}}_{\geqslant 0, <M} \to \mathbb{R}_\infty$ denote the value function such that for all ν,

$$\mathsf{Unreset}_{\mathcal{Y}}(\boldsymbol{V}_\ell)(\nu) = \boldsymbol{V}_\ell(\nu[\mathcal{Y} := 0]) \,.$$

[2] In fact, $d_\nu = M - \|\nu\|_\infty$.

- If g is a guard over $\mathcal{X}$, let $\mathsf{Guard}_g(\boldsymbol{V}_\ell) : \mathbb{R}^{\mathcal{X}}_{\geqslant 0, < M} \to \mathbb{R}_\infty$ denote the value function such that for all ν,

$$\mathsf{Guard}_g(\boldsymbol{V}_\ell)(\nu) = \begin{cases} \boldsymbol{V}_\ell(\nu) & \text{if } \nu \models g \\ -\infty & \text{if } \nu \not\models g \wedge \ell \in L_{\mathsf{Max}} \\ +\infty & \text{if } \nu \not\models g \wedge \ell \in L_{\mathsf{Min}} \,. \end{cases}$$

- If $e \in E$ is an edge from ℓ_1 to ℓ_2, let $\mathsf{PreTime}_e(\boldsymbol{V}_\ell) : \mathbb{R}^{\mathcal{X}}_{\geqslant 0, < M} \to \mathbb{R}_\infty$ denote the value function such that for all ν,

$$\mathsf{PreTime}_e(\boldsymbol{V}_{\ell_2})(\nu) = \begin{cases} \sup_{d \in [0, d_\nu)} \left[d \cdot \mathsf{wt}(\ell_1) + \mathsf{wt}(e) + \boldsymbol{V}_{\ell_1}(\nu + d) \right] & \text{if } \ell \in L_{\mathsf{Max}} \\ \inf_{d \in [0, d_\nu)} \left[d \cdot \mathsf{wt}(\ell_1) + \mathsf{wt}(e) + \boldsymbol{V}_{\ell_2}(\nu + d) \right] & \text{if } \ell \in L_{\mathsf{Min}} \,. \end{cases}$$

Then, if $\boldsymbol{V}' = \mathcal{F}(\boldsymbol{V})$, it holds that

$$\boldsymbol{V}'_\ell = \begin{cases} \boldsymbol{V}_\ell & \text{if } \ell \in L_{\mathsf{t}} \\ \max_{e=(\ell,g,\mathcal{Y},\ell')} \mathsf{PreTime}_e(\mathsf{Guard}_g(\mathsf{Unreset}_{\mathcal{Y}}(\boldsymbol{V}_{\ell'}))) & \text{if } \ell \in L_{\mathsf{Max}} \\ \min_{e=(\ell,g,\mathcal{Y},\ell')} \mathsf{PreTime}_e(\mathsf{Guard}_g(\mathsf{Unreset}_{\mathcal{Y}}(\boldsymbol{V}_{\ell'}))) & \text{if } \ell \in L_{\mathsf{Min}} \backslash L_{\mathsf{t}} \end{cases} \tag{10.2}$$

where e ranges over the edges in $\mathcal{G}$ that starts from ℓ.

Remark. The values $+\infty$ and $-\infty$ in Guards_g ensure that players cannot choose invalid delays: By the no-deadlocks assumption, from every configuration there exists a transition in $[\![\mathcal{G}]\!]$, whose value will win against $+\infty$ or $-\infty$ in (10.2).

We have detailed in Lemma 10.2 how one can perform the min and max operations over nested partitions. Let us now focus on the Guard_g and $\mathsf{Unreset}_{\mathcal{Y}}$ operations.

Lemma 10.3. *Let (P, F) be a piecewise affine value function, where P is a k-nested partition of complexity at most $\langle m \rangle$, with constants bounded by β_P in P and β_F in F. Let g be a non-diagonal guard in $\mathcal{G}$. Then there exists a piecewise affine value function (P', F'), where P' is a $(k+1)$-nested partition of complexity at most $\langle \max(m, 2n) \rangle$, with constants bounded by $\max(\beta_P, M)$ in P' and β_F in F', such that $[\![F']\!] = \mathsf{Guard}_g([\![F]\!])$.*

Proof. First, let us state that the non-diagonal guard g can be encoded as a cell $I_1 \wedge \cdots \wedge I_{2n}$, with one upper and one lower inequality for each clock. We define P' from P by partitioning each base cell by the set of affine equalities $E(I_1) \wedge \cdots \wedge E(I_{2n})$. It follows that each base cell of P' is either entirely included in g or entirely outside of it. We can thus define F' appropriately, such that $[\![F']\!] = \mathsf{Guard}_e([\![F]\!])$. Finally, the nested partition P' has the desired depth, complexity, and bounds over constants. □

Lemma 10.4. *Let (P, F) be a piecewise affine value function, where P is a k-nested partition of complexity at most $\langle m \rangle$, with constants bounded by β_P, β_F. Let $\mathcal{Y}$ be a set of clocks. Then there exists a piecewise affine value function (P', F'), where P' is a k-nested partition of complexity at most $\langle m \rangle$, with constants bounded by β_P in P' and β_F in F', such that $[\![F']\!] = \mathsf{Unreset}_{\mathcal{Y}}([\![F]\!])$.*

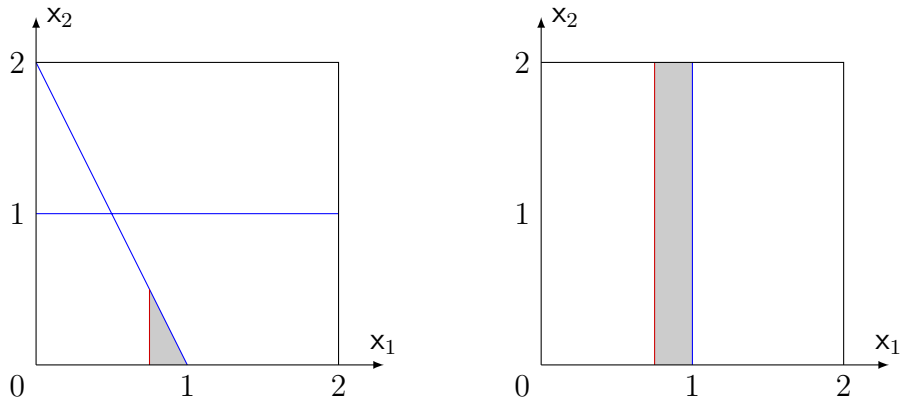


Figure 10.4.: On the left, the nested partition of Figure 10.3. On the right, the corresponding nested partition obtained by applying Lemma 10.4 for reset $\mathcal{Y} = \{\mathsf{x}_2\}$. The affine value function of the grey cell on the right is obtained from the grey cell on the left, by setting the partial derivative of x_2 to 0.

Proof. If $E : \sum_{i=1}^{n} a_i \mathsf{x}_i + b = 0$ is an affine equality, let $\mathsf{Unreset}_{\mathcal{Y}}(E)$ denote the affine equality $\sum_{i=1}^{n} a'_i \mathsf{x}_i + b = 0$, with for $i \in [1, n]$, $a'_i = 0$ if $\mathsf{x}_i \in \mathcal{Y}$ and $a'_i = a_i$ otherwise. We extend this operator to affine inequalities I in the same way, with $\mathsf{Unreset}_{\mathcal{Y}}(I)$ equal to I except for the i-th partial derivatives that is set to 0 when $\mathsf{x}_i \in \mathcal{Y}$. For each valuation $\nu \in \mathbb{R}^{\mathcal{X}}_{\geqslant 0}$, and E an affine equality (resp. inequality) $\nu \models \mathsf{Unreset}_{\mathcal{Y}}(E)$ if and only if $\nu[\mathcal{Y} := 0] \models E$. Then, if $c = I_1 \wedge \cdots \wedge I_p$ is a cell, let $\mathsf{Unreset}_{\mathcal{Y}}(c)$ denote the cell $\mathsf{Unreset}_{\mathcal{Y}}(I_1) \wedge \cdots \wedge \mathsf{Unreset}_{\mathcal{Y}}(I_p)$. It follows that $\nu \in \mathsf{Unreset}_{\mathcal{Y}}(c)$ if and only if $\nu[\mathcal{Y} := 0] \in c$. In particular, if c does not intersect the sub-space where every clock in $\mathcal{Y}$ equals 0, then $\mathsf{Unreset}_{\mathcal{Y}}(c) = \emptyset$.

Similarly, if c is a base cell of P and F maps c to the affine value function $\mathsf{y} = \sum_{i=1}^{n} a_i \mathsf{x}_i + b = 0$, let $\mathsf{Unreset}_{\mathcal{Y}}(F(c))$ denote the affine function

$$\mathsf{y} = \sum_{i=1}^{n} a'_i \mathsf{x}_i + b = 0\,,$$

with for $i \in [1, n]$, $a'_i = 0$ if $\mathsf{x}_i \in \mathcal{Y}$ and $a'_i = a_i$ otherwise. Then, for every $\nu \in \mathsf{Unreset}_{\mathcal{Y}}(c)$, it holds that

$$\boldsymbol{V}(\nu[\mathcal{Y} := 0]) = \mathsf{Unreset}_{\mathcal{Y}}(F(c))(\nu)\,.$$

If P has depth 0, it only contains one node, the root. The piecewise affine value function (P', F') with $P' = P$ and F' a partition value function mapping c_P to $\mathsf{Unreset}_{\mathcal{Y}}(F(c_P))$, encodes the function $\mathsf{Unreset}_{\mathcal{Y}}(\boldsymbol{V})$. Otherwise, we construct P' from P in the following top-down manner: Let the current node be labelled by a cell c and a set of affine equalities $\mathcal{E}$. If $\mathsf{Unreset}_{\mathcal{Y}}(c)$ is empty, we remove the current node, and its entire sub-tree, from P. Otherwise, we replace c by $\mathsf{Unreset}_{\mathcal{Y}}(c)$, and $\mathcal{E} = \{E_1, E_2, \ldots\}$ by $\{\mathsf{Unreset}_{\mathcal{Y}}(E_1), \mathsf{Unreset}_{\mathcal{Y}}(E_2), \ldots\}$. We then apply the same process recursively on the children nodes. If the current node is a leaf, and $\mathsf{Unreset}_{\mathcal{Y}}(c)$ is non-empty, we let

$F'(c) = \mathsf{Unreset}_{\mathcal{Y}}(F(c))$. The result is a nested partition P' with the desired depth, complexity, and bounds over constants, and a partition value function F' such that $[\![F']\!] = \mathsf{Unreset}_{\mathcal{Y}}([\![F]\!])$. □

An application of the $\mathsf{Unreset}_{\mathcal{Y}}$ operator is displayed in Figure 10.4.

All that is left is the $\mathsf{PreTime}_e$ operation. It is more challenging, and requires extra machinery related to *diagonal* behaviours that naturally arise when dealing with time-elapses.

10.1.3. Tubes and diagonals

An affine inequality (resp. equality) is *diagonal* if the sum of its partial derivatives is null, *i.e.* $\sum_{i=1}^{n} a_i = 0$. It follows that if ν satisfies a diagonal I then $\nu + d \models I$ for all $d \in \mathbb{R}$. A cell is called a *tube* when all of its inequalities are diagonal. When the cell is a sub-cell of some root cell c_P in a nested partition P, we relax this definition slightly, to allow for non-diagonal borders inherited from c_P.

A (k_1, k_2)-*nested tube partition* is a $(k_1 + k_2)$-nested partition, such that every node at level at most k_1 is labelled by a set of diagonal affine equalities, and thus every cell at level $k_1 + 1$ is a tube. The cells at level $k_1 + 1$ are called the *base tubes* of P. A (k_1, k_2)-nested tube partition P is said to be of complexity at most $\langle m_1, m_2 \rangle$ if for every node of P at level at most k_1, labelled by a set $\mathcal{E}$, it holds that $|\mathcal{E}| \leqslant m_1$, and for every node of P at level greater than k_1, labelled by a set $\mathcal{E}'$, it holds that $|\mathcal{E}'| \leqslant m_2$. The cells labelling nodes of P at level $k_1 + 1$ are called the base tubes of P.

Lemma 10.5. *Let (P, F) be a piecewise affine value function, where P is a k-nested partition of complexity at most $\langle m \rangle$, with constants bounded by β_P in P and β_F in F. Then there exists a piecewise affine value function (P', F'), where P' is a (k, k)-nested tube partition of complexity at most $\langle m, m \rangle$, with constants bounded by β_P in P' and β_F in F', such that $[\![F']\!] = [\![F]\!]$.*

Proof. Let P_1, P_2 be two nested partitions such that every affine equality in sets $\mathcal{E}$ of P_1 (resp. P_2) is diagonal (resp. non-diagonal), and such that $P = P_1 \oplus P_2$. One can construct them by copying the diagonal (resp. non-diagonal) borders of P in a top-down manner. The nested partition P_1 has depth at most $k - 1$, and we extend it so that every leaf node in P_1 is at level k (one can extend leaves to greater depths by constructing identical children nodes). Then, P_1 is a $(k, 0)$-nested tube partition, and P_2 is a $(0, k)$-nested tube partition. Finally, we construct P' from P_1, by placing at every base cell of P_1 labelled by a cell c a copy of P_2 restricted to the domain c. The base cells of P' are the base cells of P, and we let $F' = F$. The result is a nested partition P' with the desired depth, complexity, and bounds over constants, and a partition value function F' such that $[\![F']\!] = [\![F]\!]$. □

Given two affine equalities $E : \sum_{i=1}^{n} a_i \mathsf{x}_i + b = 0$ and $E' : \sum_{i=1}^{n} a'_i \mathsf{x}_i + b' = 0$, let $A = \sum_{i=1}^{n} a_i$ and $A' = \sum_{i=1}^{n} a'_i$ denote the sums of their respective partial derivatives.

We define their diagonal intersection as

$$E \cap_d E' : \sum_{i=1}^{n} (Aa'_i - A'a_i)\mathsf{x}_i + (Ab' - A'b) = 0\,.$$

Observe that $E \cap_d E'$ is a diagonal equality, and that $\nu \models E \wedge \nu \models E'$ implies $\nu \models E \cap_d E'$. Moreover, if E (resp. E') is diagonal then $E \cap_d E'$ is equivalent to E (resp. E'). Now, given a cell $c = I_1 \wedge \cdots \wedge I_m$ and a set $\mathcal{E}$ of affine equalities, let $\overline{\mathcal{E}}$ denote $\mathcal{E} \cup \{E(I_1), \ldots E(I_m)\}$, and let $\mathsf{Tube}(c, \mathcal{E})$ denote $\{E \cap_d E' \mid E, E' \in \overline{\mathcal{E}}\}$. The pair $(c, \mathcal{E})$ is said *atomic* if c is partitioned by $\mathsf{Tube}(c, \mathcal{E})$ in only one cell (equal to c). Intuitively, $(c, \mathcal{E})$ is atomic if the affine equalities in $\mathcal{E}$ and in the borders of c do not intersect within the smallest tube that contains c.

A (k_1, k_2)-nested tube partition is *atomic* if for every node at level greater than k_1, labelled by a cell c and a set $\mathcal{E}$, the pair $(c, \mathcal{E})$ is atomic. Intuitively, this means that in the non-diagonal part of P, the equalities that split cells into sub-cells are non-diagonal and do not intersect within their tube. Nested tube partitions can be made atomic, by introducing a bounded amount of diagonal affine equalities.

Lemma 10.6 (Lem. 3, [ABM04])**.** *Let (P, F) be a piecewise affine value function, where P is a (k_1, k_2)-nested tube partition of complexity at most $\langle m_1, m_2 \rangle$, with constants bounded by β_P in P and β_F in F. Then there exists a piecewise affine value function (P', F'), where P' is an atomic $(k_1 + 1, k_2)$-nested tube partition, of complexity at most $\langle \max(m_1, k_2 m_2^{k_2} \cdot \mathsf{Splits}(m_2, n)^{k_2}), m_2 \rangle$ and with constants bounded by $2n\beta_P^2$ in P' and β_F in F', such that $[\![F']\!] = [\![F]\!]$.*

Proof. We insert an additional level at depth $k_1 + 1$, where we will store the new diagonal equalities. We add to these new base tubes all equalities $E \cap_d E'$ derived from their sub-tree that are not equivalent to a diagonal equality in P. It is shown in [ABM04] that there are at most $m_2^{k_2}$ new diagonals for each node at level greater than k_1 in P, and thus the new base tubes are equipped with at most $k_2 m_2^{k_2} \cdot \mathsf{Splits}(m_2, n)^{k_2}$ affine equalities. The bound on constants in P' is derived from the equation describing $E \cap_d E'$. □

Example 10.3. Figure 10.5 represents the atomic nested tube partition P' associated to the nested partition P displayed in Figure 10.3. the nested tube partition P' has one diagonal level split by the 4 affine equalities in green. Each of the resulting 9 base tubes is then partitioned by the (non-diagonal) blue and red borders as in Figure 10.3. The result P' is therefore an atomic $(1, 2)$-nested tube partition of complexity $\langle 4, 2 \rangle$.

We can now compute $\mathsf{PreTime}_e(\boldsymbol{V}_{\ell'})$, with $\boldsymbol{V}_{\ell'}$ a value function encoded as a nested tube partition (P, F), and $e \in E$ an edge from ℓ to ℓ'. We will assume in the following that ℓ is a location of Max, but the case of Min is symmetrical. Let us fix a valuation $\nu \in \mathbb{R}^{\mathcal{X}}_{\geqslant 0, <M}$. Recall that

$$\mathsf{PreTime}_e([\![F]\!])(\nu) = \sup_{d \in [0, d_\nu)} \left[d \cdot \mathsf{wt}(\ell) + \mathsf{wt}(e) + [\![F]\!]_{\nu + d} \right].$$

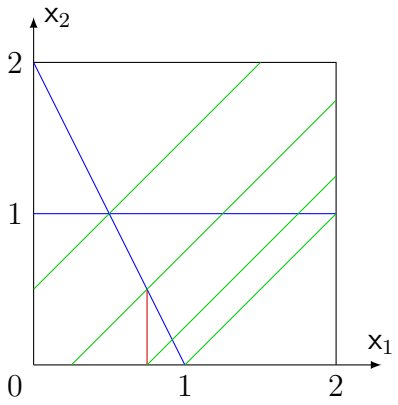


Figure 10.5.: The atomic nested tube partition derived from the nested partition of Figure 10.3 by Lemmas 10.5 and 10.6.

For every delay $d \in [0, d_\nu)$, consider the term $[\![F]\!]_{\nu+d}$. The valuations $\nu + d$ belong to a diagonal line of $\mathbb{R}^{\mathcal{X}}_{\geqslant 0}$, and range from ν to $\nu + d_\nu$. The segment $\nu + [0, d_\nu)$ intersects a finite number of base cells in P. Let $\mathcal{C}_\nu$ be the set of base cells of P reachable from ν by letting time elapse. For each cell c in $\mathcal{C}_\nu$, from the segment from ν to $\nu + d_\nu$ we isolate two delays: the infimum over delays in $\mathbb{R}_{\geqslant 0}$ such that $\nu + d \in c$, denoted d_1, and the supremum over delays such that $\nu + d \in c$, denoted d_2. As $d \mapsto [\![F]\!]_{\nu+d}$ is affine over c, so is $d \mapsto d \cdot \mathsf{wt}(\ell) + \mathsf{wt}(e) + [\![F]\!]_{\nu+d}$, and the supremum of $d \cdot \mathsf{wt}(\ell) + \mathsf{wt}(e) + [\![F]\!]_{\nu+d}$ for $\nu + d \in c$ must either be reached at (or arbitrarily close to) d_1, or at (or arbitrarily close to) d_2. Note that $\nu + d_2$ must belong to a non-diagonal border of c, while $\nu + d_1$ either belongs to a non-diagonal border of c or equals ν (whenever $\nu \in c$). Thus, the optimal value of d for evaluating the sup must correspond to either delay 0 or to a delay leading ν to a non-diagonal border (this observation is proven formally in [ABM04]).

If B is a non-diagonal border of c, and ν is a valuation of $\mathbb{R}^{\mathcal{X}}_{\geqslant 0}$, there exists a unique $d \in \mathbb{R}$ such that $\nu + d \in [\![B]\!]$. In fact, if B is described by $\sum_{i=1}^{n} a_i \mathsf{x}_i + b = 0$ and $A = \sum_{i=1}^{n} a_i$, then

$$d = -\frac{1}{A}\Big(\sum_{i=1}^{n} a_i \cdot \nu(\mathsf{x}_i) + b\Big)\,.$$

We name this delay $d_{\nu,B}$, and it must belong to $[0, d_\nu]$ if $[\![B]\!]$ is reachable from ν by time-elapse.[3] If c is a cell of $\mathcal{C}_\nu$, let $\mathcal{B}_\nu(c)$ denote the non-diagonal borders of c reachable from ν by time-elapse. The supremum $\mathsf{PreTime}_e([\![F]\!])(\nu)$ is then equal to

$$\max \begin{cases} [\![F]\!]_\nu \\ \max_{c \in \mathcal{C}_\nu} \max_{B \in \mathcal{B}_\nu(c)} [d_{\nu,B} \cdot \mathsf{wt}(\ell) + \mathsf{wt}(e) + [\![F(c)]\!]_{\nu + d_{\nu,B}}] \end{cases}$$

where $[\![F]\!]_\nu$ corresponds to the delay 0, and $[\![F(c)]\!]_{\nu+d_{\nu,B}}$ corresponds to a jump arbitrarily

[3] Note that it can be equal to d_ν, as $\mathsf{x} - M = 0$ is a border of the cell $\mathbb{R}^{\mathcal{X}}_{\geqslant 0, < M}$.

close to B.[4]

If the nested tube partition (P, F) is atomic, it follows that every other valuation in the same cell $\nu' \in [\nu]_P$ can reach the same set of borders by time elapse, *i.e.* $\mathcal{C}_\nu = \mathcal{C}_{\nu'}$ and $\mathcal{B}_\nu(c) = \mathcal{B}_{\nu'}(c)$ for all $c \in \mathcal{C}_\nu$. In consequence, we rename those sets $\mathcal{C}_{c'}$ and $\mathcal{B}_{c'}(c)$ if $c' = [\nu]_P$. We introduce an operator $\mathsf{PreTime}_{e,c,B}$, indexed by an edge, a cell and a non-diagonal border of the cell, that maps a partition value function F to the value function:

$$\nu \mapsto d_{\nu,B} \cdot \mathsf{wt}(\ell) + \mathsf{wt}(e) + [\![F(c)]\!]_{\nu + d_{\nu,B}} \ .$$

If the nested tube partition (P, F) is atomic, we can therefore write for each base cell c_b of P that $\mathsf{PreTime}_e([\![F]\!])$ restricted to domain c_b equals

$$\max([\![F]\!], \max_{c \in \mathcal{C}_{c_b}} \max_{B \in \mathcal{B}_{c_b}(c)} [\mathsf{PreTime}_{e,c,B}(F)]) \, .$$

Recall that we will ultimately compute the maximum over all edges e from ℓ to ℓ' of $\mathsf{PreTime}_e(\mathsf{Guard}_g(\mathsf{Unreset}_Y(\boldsymbol{V}_{\ell'})))$. Instead of computing the maximum with $[\![F]\!]$ in every $\mathsf{PreTime}_e$ computations, we will do it as a last step.

Lemma 10.7. *Let (P, F) be a piecewise affine value function, where P is an atomic nested tube partition, with constants bounded by β_P in P and β_F in F. Let c_b be a base cell of P, e be an edge from ℓ to ℓ', c be a cell in $\mathcal{C}_{c_b}$ and B be a border in $\mathcal{B}_{c_b}(c)$. Then there exists an affine value function f of equation $a^f_{\mathsf{y}}\mathsf{y} = \sum_{i=1}^n a^f_i \mathsf{x}_i + b^f$, with constants bounded by $\beta_P\beta_F(2n + |\mathsf{wt}(\ell)| + n|\mathsf{wt}(e)|)$, such that $f = \mathsf{PreTime}_{e,c,B}(F)$ on c_b. Moreover, $\sum_{i=1}^n a^f_i = -a^f_{\mathsf{y}} \cdot \mathsf{wt}(\ell)$.*

Proof. Let ν be a valuation in c_b. As $B \in \mathcal{B}_{c_b}(c)$ it holds that $d_{\nu,B} \in [0, d_\nu]$, such that

$$\mathsf{PreTime}_{e,c,B}(F)(\nu) = d_{\nu,B} \cdot \mathsf{wt}(\ell) + \mathsf{wt}(e) + [\![F(c)]\!]_{\nu + d_{\nu,B}} \ .$$

Let $a_{\mathsf{y}}\mathsf{y} = \sum_{i=1}^n a_i\mathsf{x}_i + b$ be the equation of $F(c)$, and let $A = \sum_{i=1}^n a_i$. Let $\sum_{i=1}^n a'_i\mathsf{x}_i + b' = 0$ be the equation of B, with $A' = \sum_{i=1}^n a'_i \neq 0$ the sum of its partial derivatives. We obtain the following equalities:

$$\begin{aligned}
d_{\nu,B} &= -\frac{1}{A'}\left(\textstyle\sum_{i=1}^n a'_i \cdot \nu(\mathsf{x}_i) + b'\right) \\
[\![F(c)]\!]_{\nu + d_{\nu,B}} &= \frac{1}{a_{\mathsf{y}}}\left[\textstyle\sum_{i=1}^n a_i \cdot \nu(\mathsf{x}_i) + A d_{\nu,B} + b\right] \\
[\![F(c)]\!]_{\nu + d_{\nu,B}} \cdot A' a_{\mathsf{y}} &= \textstyle\sum_{i=1}^n (A' a_i - A a'_i) \cdot \nu(\mathsf{x}_i) + (A'b - Ab') \\
\mathsf{PreTime}_{e,c,B}(F)(\nu) \cdot A' a_{\mathsf{y}} &= \textstyle\sum_{i=1}^n (A' a_i - A a'_i - a_{\mathsf{y}} a'_i \cdot \mathsf{wt}(\ell)) \cdot \nu(\mathsf{x}_i) \\
&\quad + (A'b - Ab' - a_{\mathsf{y}} b' \cdot \mathsf{wt}(\ell) + A' a_{\mathsf{y}} \cdot \mathsf{wt}(e)) \, ,
\end{aligned}$$

and thus $\mathsf{PreTime}_{e,c,B}(F)(\nu)$ is described by an equation $a^f_{\mathsf{y}}\mathsf{y} = \sum_{i=1}^n a^f_i \mathsf{x}_i + b^f$, with $a^f_{\mathsf{y}} \in \mathbb{N}_{>0}$, $a^f_{\mathsf{y}} \leqslant n\beta_P\beta_F$, $b^f \in \mathbb{Z}$, $|b^f| \leqslant \beta_P\beta_F(2n + |\mathsf{wt}(\ell)| + n|\mathsf{wt}(e)|)$, and for all $i \in [1, n]$, $a^f_i \in \mathbb{Z}$ and $|a^f_i| \leqslant \beta_P\beta_F(2n + |\mathsf{wt}(\ell)|)$.

[4] In particular, if $\nu + d_{\nu,B} \notin c$ then $F(c)$ evaluated on $\nu + d_{\nu,B}$ may not equal $[\![F]\!]_{\nu + d_{\nu,B}}$.

Finally, $\sum_{i=1}^{n} a_i^f = -a_{\mathsf{y}}^f \cdot \mathsf{wt}(\ell)$ because $\sum_{i=1}^{n}(A'a_i - Aa_i') = 0$. □

We can use the observation that $\sum_{i=1}^{n} a_i^f = -a_{\mathsf{y}}^f \cdot \mathsf{wt}(\ell)$ holds on the output of Lemma 10.7 to specialize Lemma 10.2 on such instances, so as to obtain a lower output complexity.

Lemma 10.8. *Let (P_i, F_i), $i = 1, \ldots, q$ be q piecewise affine value functions, where each P_i is a (k_1, k_2)-nested tube partition of complexity at most $\langle m_1, m_2\rangle$, with constants bounded by β_P in P_i and β_F in F_i. Suppose that they share the same partition, i.e. $P_1 = \cdots = P_q = P$, and that for all affine value functions mapped to base cells of P by an F_i, of equation $a_{\mathsf{y}}\mathsf{y} = \sum_{i=1}^{n} a_i\mathsf{x}_i + b$, it holds that $\sum_{i=1}^{n} a_i = -\mathsf{wt}(\ell) \cdot a_{\mathsf{y}}$. Then there exists a piecewise affine value function (P', F') of domain c_P, where P' is a $(k_1 + 1, k_2)$-nested tube partition of complexity at most $\langle \max(m_1, q^2), m_2\rangle$, with constants bounded by $2\beta_P^2$ in P' and β_F in F', such that $[\![F']\!] = \min_{i=1,\ldots,q} [\![F_i]\!]$ (resp. $[\![F']\!] = \max_{i=1,\ldots,q} [\![F_i]\!]$).*

Proof. The construction follows the proof of Lemma 10.2, with the additional observation that all intersections $E \cap_y E'$ must be diagonal equalities in $\mathbb{R}^{\mathcal{X}}_{\geqslant 0}$, since $\sum_{i=1}^{n}(a_{\mathsf{y}}a_i' - a_{\mathsf{y}}'a_i) = -a_{\mathsf{y}}a_{\mathsf{y}}' \cdot \mathsf{wt}(\ell) + a_{\mathsf{y}}'a_{\mathsf{y}} \cdot \mathsf{wt}(\ell) = 0$. Therefore, the new equalities are all diagonal, and the new level of nodes in P' is added at level $k_1 + 1$. As the inputs share the same nested partition the step $P_1 \oplus \cdots \oplus P_q$ can be skipped, letting us obtain the desired complexity. □

When the input mappings do not have this property, we will rely on a weaker result, derived from Lemma 10.2 directly: the new borders may be diagonal, or they may be non-diagonal, so we add a new level for each possibility.

Corollary 10.1. *Let (P_i, F_i), $i = 1, \ldots, q$ be q piecewise affine value functions over the same domain c_P, where each P_i is a (k_1, k_2)-nested tube partition of complexity at most $\langle m_1, m_2\rangle$, with constants bounded by β_P in P_i and β_F in F_i. Then there exists a piecewise affine value function (P', F') of domain c_P, where P' is a $(k_1 + 1, k_2 + 1)$-nested tube partition of complexity at most $\langle \max(m_1 q, q^2), \max(m_2 q, q^2)\rangle$, with constants bounded by $2\beta_P^2$ in P' and β_F in F', such that $[\![F']\!] = \min_{i=1,\ldots,q} [\![F_i]\!]$ (resp. $[\![F']\!] = \max_{i=1,\ldots,q} [\![F_i]\!]$).*

We can now use Lemmas 10.7 and 10.8 to implement $\mathsf{PreTime}_e$ over nested tube partitions.

Lemma 10.9. *Let (P, F) be a piecewise affine value function, where P is an atomic (k_1, k_2)-nested tube partition of complexity at most $\langle m_1, m_2\rangle$, with constants bounded by β_P in P and β_F in F. Let e be an edge from ℓ to ℓ'. Then there exists a piecewise affine value function (P', F'), where P' is a $(k_1 + 1, k_2)$-nested tube partition of complexity at most $\langle \max(m_1, 4k_2^2 \cdot \mathsf{Splits}(m_2, n)^{2k_2}), m_2\rangle$, with constants bounded by $2\beta_P^2$ in P' and $\beta_P\beta_F(2n + |\mathsf{wt}(\ell)| + n|\mathsf{wt}(e)|)$ in F', such that for each base cell c_b of P, $[\![F']\!] = \max_{c \in \mathcal{C}_{c_b}} \max_{B \in \mathcal{B}_{c_b}(c)} [\mathsf{PreTime}_{e,c,B}(F)]$.*

Proof. For each base cell c_b of P, we can compute (P'_{c_b}, F'_{cb}), encoding the value function $\max_{c \in \mathcal{C}_{c_b}} \max_{B \in \mathcal{B}_{c_b}(c)} [\mathsf{PreTime}_{e,c,B}(F)]$ over c_b, with Lemmas 10.7 and 10.8. There are at most $\max(1, k_2\mathsf{Splits}(m_2, n)^{k_2})$ cells in $\mathcal{C}_{c_b}$ and at most 2 borders in $\mathcal{B}_{c_b}(c)$, such

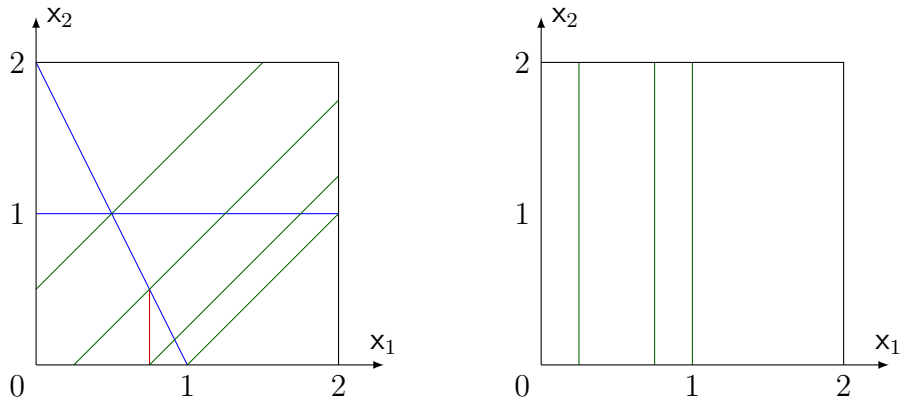


Figure 10.6.: On the left, the atomic nested tube partition of Figure 10.5. On the right, the corresponding nested tube partition obtained by applying Corollary 10.2 for reset $\mathcal{Y} = \{\mathsf{x}_2\}$. Note that every border on the right is derived from a diagonal border on the left, by atomicity, such that the blue and red borders can be ignored.

that by Lemma 10.8, (P'_{c_b}, F'_{cb}) is a $(k_1 + 1, k_2)$-nested tube partition of complexity at most $\langle \max(m_1, 4k_2^2 \cdot \mathsf{Splits}(m_2, n)^{2k_2}), m_2 \rangle$, with constants bounded by $2\beta_P^2$ in P'_{c_b} and $\beta_P \beta_F (2n + |\mathsf{wt}(\ell)| + n|\mathsf{wt}(e)|)$ in F'_{cb}. Then, we can use all (P'_{c_b}, F'_{cb}) to define a unique (P', F'), by inserting all of the new diagonal equalities at level $k_1 + 1$ (the base tubes of P). □

We finally need to adapt Lemmas 10.3 and 10.4 to handle nested tube partitions instead of nested partitions.

Corollary 10.2. *If (P, F) is a (k_1, k_2)-nested tube partition of complexity at most $\langle m_1, m_2 \rangle$, then applying the Guard_g operator for a non-diagonal guard g results in a $(k_1, k_2 + 1)$-nested tube partition of complexity at most $\langle m_1, \max(m_2, 2n) \rangle$. Applying the $\mathsf{Unreset}_{\mathcal{Y}}$ operator results in a $(k_1, \max(k_1, k_2))$-nested tube partition of complexity at most $\langle m_1, \max(m_1, m_2) \rangle$ when (P, F) is atomic.*

Indeed, if (P, F) is atomic and $\mathcal{Y} \neq \emptyset$ then the unreset of a border B of a base cell c of P can only partition $\mathsf{Unreset}_{\mathcal{Y}}(c)$ into several sub-cells if B is diagonal, therefore the resulting partition can only contain the $\mathsf{Unreset}$ of diagonal borders. Note that $\mathsf{Unreset}_{\mathcal{Y}}(B)$ may be diagonal or not, depending on what clocks are reset, and thus we obtain a complexity of $\langle m_1, m_1 \rangle$. If $\mathcal{Y} = \emptyset$ the complexity stays at $\langle m_1, m_2 \rangle$. Let us now bring everything together for the value iteration operator $\mathcal{F}$.

Proposition 10.1. *Let $\boldsymbol{V}$ be a piecewise affine value function, encoded as a nested tube partition (P_ℓ, F_ℓ) for each $\ell \in L$, where every P_ℓ is an atomic (k_1, k_2)-nested tube partition of complexity at most $\langle m_1, m_2 \rangle$, with constants bounded by β_P in P_ℓ and β_F in F_ℓ. Let $q = |E|$ be the number of edges in $\mathcal{G}$. Then there exists a piecewise affine value function $\boldsymbol{V}'$, encoded as a nested tube partition (P'_ℓ, F'_ℓ) for each $\ell \in L$, where every P'_ℓ is*

an atomic $(k_1+5, \max(k_1,k_2)+3)$*-nested tube partition of complexity at most* $\langle m_1', m_2'\rangle$ *and constants bounded by* β_P' *in* P_ℓ' *and* β_F' *in* F_ℓ'*, such that* $\boldsymbol{V}' = \mathcal{F}(\boldsymbol{V})$*. If we let* $\mathbf{k}$ *denote* $\max(k_1,k_2)$ *and* $\mathbf{m}$ *denote* $\max(m_1,m_2)$*, it holds that:*

$$\begin{array}{rcl} m_1' & \leqslant & \max \begin{cases} m_1, \\ 2q^2, \\ 8q(\mathbf{k}+1)^2 \cdot \mathsf{Splits}(\max(\mathbf{m},2n),n)^{2(\mathbf{k}+1)}, \\ (\mathbf{k}+3)(\max(4,2q\mathbf{m},4qn))^{\mathbf{k}+3} \cdot \mathsf{Splits}(\max(4,2q\mathbf{m},4qn),n)^{\mathbf{k}+3} \end{cases} \\ m_2' & \leqslant & \max(2q\mathbf{m}, 4qn, 2q^2) \\ \beta_P' & \leqslant & 2^{23} n^{17} \max(\beta_P, M)^{32} \\ \beta_F' & \leqslant & 2n\beta_F \max(\beta_P, M)(2n + w^L_{\max} + n w^E_{\max}) \end{array}$$

Proof. Fix a location $\ell \in L_{\mathsf{Max}}$. the case L_{Min} is symmetrical and will not be detailed. If $\ell \in L_{\mathsf{t}}$, let $(P_\ell', F_\ell') = (P_\ell, F_\ell)$. Otherwise,

$$\boldsymbol{V}_\ell' = \max_{e=(\ell,g,\mathcal{Y},\ell')} \mathsf{PreTime}_e(\mathsf{Guard}_g(\mathsf{Unreset}_{\mathcal{Y}}(\boldsymbol{V}_{\ell'}))) \,.$$

For every location ℓ', we construct, by Corollary 10.2 and Lemma 10.6, a nested tube partition $(P_{\ell'}'', F_{\ell'}'')$ encoding $\mathsf{Guard}_g(\mathsf{Unreset}_{\mathcal{Y}}(\boldsymbol{V}_{\ell'}))$. Every $P_{\ell'}''$ is an atomic $(k_1+1, \mathbf{k}+1)$-nested tube partition of complexity at most $\langle m_1'', m_2''\rangle$, with

$$m_1'' \leqslant \mathbf{k}(\max(\mathbf{m},2n))^{\mathbf{k}} \cdot \mathsf{Splits}(\max(\mathbf{m},2n),n)^{\mathbf{k}}$$

and $m_2'' \leqslant \max(\mathbf{m},2n)$, with constants bounded by $2n\max(\beta_P,M)^2$ in $P_{\ell'}''$ and β_F in $F_{\ell'}''$.

The next step is to apply Lemma 10.9 in order to obtain for every edge from ℓ to ℓ' a pair (P_e'', F_e'') such that $\max(\llbracket F_{\ell'}'' \rrbracket, \llbracket F_e'' \rrbracket) = \mathsf{PreTime}_e(\llbracket F_{\ell'}'' \rrbracket)$. The nested partition P_e'' is a $(k_1+2, \mathbf{k}+1)$-nested tube partition of complexity at most $\langle m_1''', \max(\mathbf{m},2n)\rangle$, with

$$m_1''' \leqslant \max(m_1'', 4(\mathbf{k}+1)^2 \cdot \mathsf{Splits}(\max(\mathbf{m},2n),n)^{2(\mathbf{k}+1)}) \,,$$

with constants bounded by $4n^2\max(\beta_P,M)^4$ in P_e'' and $2n\beta_F\max(\beta_P,M)(2n + w^L_{\max} + nw^E_{\max})$ in F_e''.

We can now apply Corollary 10.1 to compute $\max_{e=(\ell,g,\mathcal{Y},\ell')} \llbracket F_e'' \rrbracket$.

The result is a $(k_1+3, \mathbf{k}+2)$-nested tube partition (P_E'', F_E'') of complexity at most

$$\langle \max(qm_1'', 4q(\mathbf{k}+1)^2 \cdot \mathsf{Splits}(\max(\mathbf{m},2n),n)^{2(\mathbf{k}+1)}, q^2), \max(q\mathbf{m}, 2qn, q^2)\rangle \,,$$

with constants bounded by $32n^4\max(\beta_P,M)^8$ in P_E'' and $2n\beta_F\max(\beta_P,M)(2n + w^L_{\max} + nw^E_{\max})$ in F_E''. Finally, we apply Corollary 10.1 and Lemma 10.6 to compute an atomic representation of $\max(\llbracket F_E'' \rrbracket, \llbracket F_{\ell'}'' \rrbracket)$, resulting in a nested partition encoding $\boldsymbol{V}_\ell'$ with the desired bounds on depth, complexity, and greatest constants. □

10.1.4. Exponential vs doubly-exponential

We can now state the main result of this chapter.

Theorem 10.1. *Given $i \geqslant 0$, computing $\mathsf{Val}^i_{\mathcal{G}}$ can be done in time doubly-exponential in i and exponential in the size of $\mathcal{G}$.*

Proof. This result is derived from Proposition 10.1. Indeed, we can apply it i times, and obtain a piecewise affine value function of depth at most linear in k_1, k_2, and i, thus of depth linear in the size of the input value function and in i. The final complexity $\langle m'_1, m'_2 \rangle$, and the bounds on constants β'_P and β'_F, are at most doubly exponential in i and exponential in the size of $\mathcal{G}$, and we conclude by Lemma 10.1. We have seen that the size of the piecewise affine value functions handled in the computation is doubly-exponentially bounded. This translates into a 2-EXPTIME procedure as the operations over value functions that we introduced can all be performed in time polynomial relative to the size of their input. □

Let us now discuss the result of [ABM04], where an exponential upper bound on the bounded value problem is obtained with a non-symbolic algorithm on a slightly different setting (with non-negative weights only, without final weights, in a concurrent setting).

Their concurrent setting generalizes ours, the sign of weights had seemingly no impact in the proofs, and the symbolic version requires minor changes related to the continuity of value functions and to the way guards are handled. These changes should not affect complexity significantly, and the reason for this exponential gap is not apparent.

As a tentative answer, we make the following observation. If the game has no resets, *i.e.* $\mathcal{Y} = \emptyset$ on all edges, the complexity of our approach becomes exponential.[5] In [ABM04], the way one should deal with resets is not detailed, it is therefore left open whether we could obtain an exponential bound or whether their solution is in fact doubly-exponential.

We claim that the bound obtained in Proposition 10.1 is tight, in the sense that there exists a nested tube partition P, of complexity $\langle m_1, m_2 \rangle$, such that if P' of complexity $\langle m'_1, m'_2 \rangle$ is obtained after applying $\mathcal{F}$, then $m'_1 = \Theta(m_1^{n-1})$. This is the root of the issue, as we would need $m'_1 = \mathcal{O}(m_1)$ in order to obtain an exponential bound:

- If a transformation $m' := am$ is applied i times on some m_0, for a fixed constant a, we obtain $a^i m_0$, which is exponential in i.
- However, applying i times the transformation $m' := m^{n-1}$ on m_0 outputs $m_0^{(n-1)^i}$, which is doubly exponential in i.

Example 10.4. Let $p_1, \dots p_{n-1}$ be $n-1$ pairwise distinct prime numbers. For every $d \in [1, n-1]$, $i \in [1, p_d - 1]$, let E^i_d denote the affine equality

$$\mathsf{x}_d - \mathsf{x}_n - \frac{i}{p_d} = 0\,,$$

[5] In this case, the $\mathsf{Unreset}_{\mathcal{Y}}$ steps can be skipped, and one can replace every $\mathbf{m}$ and $\mathbf{k}$ by m_2 and k_2, respectively, in Proposition 10.1. Overall, m'_1 (resp. m'_2) end up linear in m_1 (resp. m_2) instead of being polynomial.

and let $\mathcal{E}_d = \{E_d^i \mid i \in [1, p_d - 1]\}$. Let $\mathcal{E}$ denote $\mathcal{E}_1 \cup \cdots \cup \mathcal{E}_{n-1}$. It holds that

$$|\mathcal{E}| = \sum_{d=1}^{n-1} (p_d - 1)\,,$$

because the affine equalities are pairwise distinct: $E_d^i = E_{d'}^{i'}$ implies $d = d'$ and $\frac{i}{p_d} = \frac{i'}{p'_d}$, where $\frac{i}{p_d}$ and $\frac{i'}{p'_d}$ are irreducible fractions, such that $i = i'$.

Let P denote the 1-nested partition of domain $\mathbb{R}^{\mathcal{X}}_{\geqslant 0, < M}$ where the root is partitioned by $\mathcal{E}$. Since every E_d^i is diagonal, P is in fact an atomic $(1, 0)$-nested tube partition of complexity $\langle \sum_{d=1}^{n-1}(p_d - 1), 0\rangle$.

After applying $\mathsf{Unreset}_{\{\mathsf{x}_n\}}$, we obtain a $(0, 1)$-nested tube partition of complexity $\langle 0, \sum_{d=1}^{n-1}(p_d - 1)\rangle$. Indeed, the root is now split by the set of non-diagonal affine equalities

$$\mathcal{E}' = \{\mathsf{Unreset}_{\{\mathsf{x}_n\}}(E_d^i) : \mathsf{x}_d - \frac{i}{p_d} = 0 \mid d \in [1, n-1], i \in [1, p_d - 1]\}\,,$$

with $|\mathcal{E}'| = \sum_{d=1}^{n-1}(p_d - 1)$ once again.

The next step is to apply Lemma 10.6 in order to make the tube partition atomic. We claim that the output is a $(1, 1)$-nested tube partition with $\prod_{d=1}^{n-1}(p_d - 1)$ diagonal equalities. This comes from the fact that every new diagonal intersection $\mathsf{Unreset}_{\{\mathsf{x}_n\}}(E_d^i) \cap_d \mathsf{Unreset}_{\{\mathsf{x}_n\}}(E_{d'}^{i'})$, with $d \neq d'$, is unique: its equation is

$$\mathsf{x}_{d'} - \mathsf{x}_d + \frac{i}{p_d} - \frac{i'}{p_{d'}} = 0\,,$$

such that if E_d^j and $E_{d'}^{j'}$ create the same diagonal, then

$$\frac{i - j}{p_d} = \frac{i' - j'}{p_{d'}}\,,$$

and thus $i = j$ and $i' = j'$ (they are both irreducible fractions with distinct denominators).

Overall, we started with $m_1 = \sum_{d=1}^{n-1}(p_d - 1)$ diagonal borders, and ended up with $m'_1 = \prod_{d=1}^{n-1}(p_d - 1)$ diagonal borders after the unreset and pretime steps. It is well known (see *e.g.* [Ros41]) that for $\gamma \in \mathbb{N}$ large enough, if p is the γ-th prime number, then $\gamma \log(\gamma) + \gamma \log(\log(\gamma)) - \gamma < p - 1 < \gamma \log(\gamma) + \gamma \log(\log(\gamma))$. It follows that for all $\gamma \in \mathbb{N}$ large enough, if we fix p_1 as the γ-th prime number, and $p_2, \ldots, p_{n-1}$ the next prime numbers in order, then

$$\begin{gathered}\gamma \log(\gamma) + \gamma \log(\log(\gamma)) - \gamma < p_1 - 1\,,\\ p_{n-1} - 1 < (\gamma + n - 1)\log(\gamma + n - 1) + (\gamma + n - 1)\log(\log(\gamma + n - 1))\,.\end{gathered}$$

Overall, $m_1 = \Theta(\gamma \log(\gamma))$ and $m'_1 = \Theta((\gamma \log(\gamma))^{n-1})$, thus our operations increase m_1 polynomially, whereas a linear increase is needed.

It should be noted that we do not know of any example where a double-exponential

splitting of the state-space is required, and a finer analysis of our procedure may still be able to match the results claimed in [ABM04]. In particular, the construction of Example 10.4 cannot *a priori* be iterated recursively, as the output diagonals do not have the same form as the input ones.

10.1.5. Bounding partial derivatives

In the previous analysis, we explained that constants (partial derivatives and additive constants) grow polynomially at each elementary step, which is enough for an exponential upper bound (double-exponential growth of their value, stored in binary). This rough analysis will not be fine enough for some of our results, in particular the approximation results of Chapter 11 will be sensitive to the partial derivatives in a linear (and not logarithmic) way. In this section, we study the growth of these partial derivatives more closely. This time, our focus will not be on the space required to store affine equations, but rather on mathematical properties of the value functions, namely their Lipschitz-continuity, closely related to bounds on partial derivatives. As a result, we revert to denoting affine equations as terms $\mathsf{y} = \sum_{i=1}^{n} a_i \mathsf{x}_i + b$ with rational constants instead of using integers with a separately stored denominator a_{y}.

Definition 10.4. The function wt_{t} is said to be Λ-*Lipschitz-continuous* when $|\mathsf{wt}_{\mathsf{t}}(s,\nu) - \mathsf{wt}_{\mathsf{t}}(s,\nu')| \leqslant \Lambda\|\nu - \nu'\|_\infty$ for all valuations ν, ν', where $\|\nu\|_\infty = \max_{\mathsf{x}\in\mathcal{X}} |\nu(\mathsf{x})|$ is the ∞-norm of vector $\nu \in \mathbb{R}^{\mathcal{X}}$. The function wt_{t} is said to be Lipschitz-continuous if it is Λ-Lipschitz-continuous, for some Λ.

Since final weight functions are piecewise affine and continuous on regions, they are Λ-Lipschitz-continuous, for a given constant $\Lambda \geqslant 0$.

We will maintain as an invariant that V_ℓ^i is Lipschitz-continuous over each region and for all ℓ:

Lemma 10.10. *If every final weight in a WTG $\mathcal{G}$ is Λ-Lipschitz-continuous on regions (and piecewise affine), then $\mathsf{Val}_{\mathcal{G}}^i$ is $\Lambda\Lambda'$-Lipschitz-continuous on regions, with Λ' polynomial in $w_{\max}^L$ and $|\mathcal{X}|$, and exponential in i.*

Note that for a piecewise affine function with finitely many pieces, being Λ-Lipschitz-continuous on regions is equivalent to being continuous on regions and having all partial derivatives bounded by Λ in absolute value. The rest of this section is dedicated to proving Lemma 10.10.

Lemma 10.11. *If for all $\ell \in L$, $\boldsymbol{V}_\ell$ is piecewise affine with finitely many pieces that have all their partial derivatives bounded by Λ in absolute value, then for all $\ell \in L$, $\mathcal{F}(\boldsymbol{V})_\ell$ is continuous on regions and piecewise affine with partial derivatives bounded by $\max(\Lambda, |\mathsf{wt}(\ell)| + (n-1)\Lambda)$ in absolute value.*

Proof. We will show that for every region r, $\mathcal{F}(\boldsymbol{V})$ restricted to r has those properties. Note that they are transmitted over finite min and max operations. The continuity on regions is easy to prove because it is stable by inf and sup. There exists a partition

cost function (P_ℓ, F_ℓ) for each $\ell \in L$ that represents $\boldsymbol{V}$. As explained before, a crucial property is that, for a given valuation ν, the delays d that need to be considered in the sup or inf operation of $\mathcal{F}(\boldsymbol{V})_\ell(\nu)$ correspond to the intersection points of the diagonal half line containing the time successors of ν and borders of cells (if ν^b is such a valuation, $d = \|\nu^b - \nu\|_\infty$ is the associated delay). In particular, there is a finite number of such borders, and the final $\mathcal{F}(\boldsymbol{V})_\ell$ function can be written as a finite nesting of finite min and max operations over affine terms, each corresponding to a choice of delay and an edge to take. Formally, there are several cases to consider to define those terms, depending on delay and edge choices. For each available edge e, those terms can either be:

1. If a delay 0 is taken and all clocks in $\mathcal{Y} \subseteq \mathcal{X}$ are reset by e, then

$$\mathsf{wt}_\Sigma((\ell,\nu) \xrightarrow{0} (\ell,\nu) \xrightarrow{e} (\ell',\nu[\mathcal{Y}:=0])) = \mathsf{wt}_\Sigma(e) + \boldsymbol{V}_{\ell'}(\nu[\mathcal{Y}:=0])$$

2. If a delay $d > 0$ (leading to valuation ν^b on border B) is taken and all clocks in $\mathcal{Y} \subseteq \mathcal{X}$ are reset by e, then

$$\mathsf{wt}_\Sigma((\ell,\nu) \xrightarrow{d} (\ell,\nu^b) \xrightarrow{e} (\ell',\nu^b[\mathcal{Y}:=0])) = \mathsf{wt}_\Sigma(\ell)\cdot d + \mathsf{wt}_\Sigma(e) + \boldsymbol{V}_{\ell'}(\nu^b[\mathcal{Y}:=0])$$

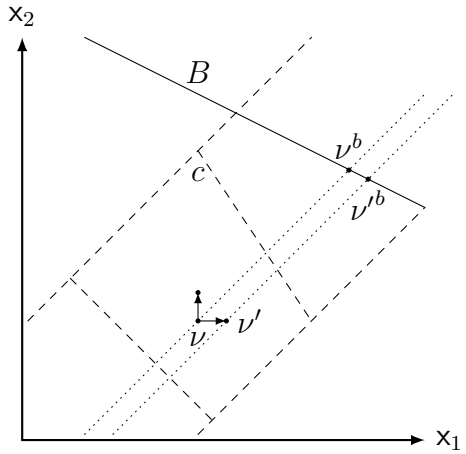


Figure 10.7.: A cell c as described in the proof of Lemma 10.11. Dashed lines are borders of c, dotted lines are proof constructions.

In the first case, the resulting partial derivatives are 0 for clocks in $\mathcal{Y}$, and the same as the partial derivatives in $\boldsymbol{V}_{\ell'}$ for all other clocks, which allows us to conclude that they are bounded by Λ. We now consider the second case. We argue that the second case could be decomposed as a delay followed by an edge of the first case, meaning that we can assume $\mathcal{Y} = \emptyset$ without loss of generality.

There are again two cases: the border B being inside a region or on the frontier of a region.

If the border is not the frontier of a region, it is the intersection points of two affine pieces of $\boldsymbol{V}_{\ell'}$ whose equations (in the space $\mathbb{R}^{n+1}$ whose n first coordinates are the clocks

$(\mathsf{x}_1, \ldots, \mathsf{x}_n)$ and the last coordinate y corresponds to the value $\boldsymbol{V}_{\ell'}(\mathsf{x}_1, \ldots, \mathsf{x}_n)$) can be written $y = \sum_{i=1}^n a_i \mathsf{x}_i + b$ (before the border) and $y = \sum_{i=1}^n a'_i \mathsf{x}_i + b'$ (after the border). Therefore, valuations at the borders all fulfill the equation

$$\sum_{i=1}^{n} (a'_i - a_i)\mathsf{x}_i + b' - b = 0 \tag{10.3}$$

We let $A = \sum_{i=1}^n (a'_i - a_i)$. Consider that ℓ is a location of Min (the very same reasoning applies to the case of a location of Max). Since $\mathcal{F}$ computes an infimum, we know that the function mapping the delay d to the weight obtained from reaching $\nu + d$ is decreasing before the border and increasing after. These functions are locally affine which implies that their slopes verify:

$$\mathsf{wt}(\ell) + \sum_{i=1}^{n} a_i \leqslant 0 \quad \text{and} \quad \mathsf{wt}(\ell) + \sum_{i=1}^{n} a'_i \geqslant 0\,. \tag{10.4}$$

We deduce from these two inequalities that $A \geqslant 0$. The case where $A = 0$ would correspond to the case where the border contains a diagonal line, which is forbidden, and thus $A > 0$. Consider now a valuation of coordinates $\nu = (\mathsf{x}_1, \ldots, \mathsf{x}_n)$ and another valuation of coordinates $\nu' = (\mathsf{x}_1, \ldots, \mathsf{x}_{k-1}, \mathsf{x}_k + \lambda, \mathsf{x}_{k+1}, \ldots, \mathsf{x}_n)$. The delays d and d' needed to arrive to the border starting from these two valuations are such that $\nu + d$ and $\nu' + d'$ both verify (10.3). We can then deduce

$$d' - d = \lambda \frac{a_k - a'_k}{A}\,.$$

It is now possible to compute the partial derivative of $\mathcal{F}(\boldsymbol{V})_\ell$ in the k-th coordinate using

$$\frac{\mathcal{F}(\boldsymbol{V})_{\ell,\nu'} - \mathcal{F}(\boldsymbol{V})_{\ell,\nu}}{\lambda} = \frac{\mathsf{wt}(\ell)(d' - d) + \boldsymbol{V}_{\ell',\nu'+d'} - \boldsymbol{V}_{\ell',\nu+d}}{\lambda} \tag{10.5}$$

. We may compute it by using the equations of the affine pieces before or after the border. We thus obtain

$$\frac{\mathcal{F}(\boldsymbol{V})_{\ell,\nu'} - \mathcal{F}(\boldsymbol{V})_{\ell,\nu}}{\lambda} = \frac{a_k - a'_k}{A}\left(\mathsf{wt}(\ell) + \sum_{i=1}^{n} a_i\right) + a_k$$
$$\frac{\mathcal{F}(\boldsymbol{V})_{\ell,\nu'} - \mathcal{F}(\boldsymbol{V})_{\ell,\nu}}{\lambda} = \frac{a_k - a'_k}{A}\left(\mathsf{wt}(\ell) + \sum_{i=1}^{n} a'_i\right) + a'_k$$

In the case where $a_k \geqslant a'_k$, the first equation, with (10.4), allows us to obtain that the partial derivative is at most a_k. We may then lower $\mathsf{wt}(\ell)$ by $-\sum_{i=1}^n a'_i$ to obtain that the partial derivative is at least a'_k. Since a_k and a'_k are bounded in absolute value by Λ, so is the partial derivative. We get the same result by reasoning on the second equation if $a'_k \geqslant a_k$.

We now come back to the case where the border is on the frontier of a region. Then,

it is a segment of a line of equation $\mathsf{x}_k = c$ for some k and c. $\boldsymbol{V}_{\ell'}$ contains at most three values for points of B: the limit coming from before the border, the value at the border, and the limit coming from after the border. The computation of $\mathcal{F}(\boldsymbol{V})$ considers values obtained from all three and takes the min (or the max).

Now, let $y = \sum_{i=1}^{n} a_i \mathsf{x}_i + b$ be the equation defining the affine piece of $\boldsymbol{V}_{\ell'}$ before the border (resp. at the border, after the border). Consider a valuation of coordinates $\nu = (\mathsf{x}_1, \ldots, \mathsf{x}_n)$ and another valuation of coordinates $\nu' = (\mathsf{x}_1, \ldots, \mathsf{x}_{j-1}, \mathsf{x}_j + \lambda, \mathsf{x}_{j+1}, \ldots, \mathsf{x}_n)$. The delays d and d' needed to arrive to the border starting from these two valuations are such that $\nu + d$ and $\nu' + d'$ both verify $\mathsf{x}_k = c$. We can then deduce that $d' - d = 0$ if $j \neq k$ and $d' - d = -\lambda$ if $j = k$. It is now possible to compute the partial derivative of $\mathcal{F}(\boldsymbol{V})_\ell$ in the j-th coordinate using (10.5) again. We may compute it by using the equations of the affine piece before the border (resp. at the border, after the border). Then,

$$\boldsymbol{V}_{\ell',\nu+d} = \sum_{i=1}^{n} a_i(\mathsf{x}_i + d) + b = \left(\sum_{i=1, i\neq k}^{n} a_i(\mathsf{x}_i + d) \right) + a_k c + b$$
$$\boldsymbol{V}_{\ell',\nu'+d'} = \left(\sum_{i=1, i\neq k}^{n} a_i(\mathsf{x}_i + d') \right) + a_k c + b\,.$$

We thus obtain

$$\frac{\mathcal{F}(\boldsymbol{V})_{\ell,\nu'} - \mathcal{F}(\boldsymbol{V})_{\ell,\nu}}{\lambda} = a_j \text{ if } j \neq k$$
$$\frac{\mathcal{F}(\boldsymbol{V})_{\ell,\nu'} - \mathcal{F}(\boldsymbol{V})_{\ell,\nu}}{\lambda} = -\mathsf{wt}(\ell) - \sum_{i=1, i\neq k}^{n} a_i \text{ otherwise}$$

Then, the partial derivatives are bounded, in absolute value, by $|\mathsf{wt}(\ell)| + (n-1)\Lambda$. □

As a corollary, we obtain Lemma 10.10.

10.2. Divergent weighted timed games

We will now explain how to compute all values in a divergent weighted timed game $\mathcal{G}$. This is a decidability result, and our computations will be performed on $\mathcal{R}(\mathcal{G})$. First, we use the continuity on regions of final weights to argue that final weights can be considered finite without loss of generality: indeed, the region abstraction can be seen as a (finite) reachability two-player game by saying that (ℓ, r) belongs to Min (resp. Max) if $\ell \in L_{\mathsf{Min}}$ (resp. $\ell \in L_{\mathsf{Max}}$). If (ℓ, ν) is a target configuration such that $\mathsf{Val}(\ell, \nu) = +\infty$, then for all $\nu' \in [\nu], \mathsf{Val}(\ell, \nu') = +\infty$. Therefore, the attractor of Max to these $+\infty$ target configurations in $[\![\mathcal{G}]\!]$ is a set of regions, equal to the attractor of Max to these $+\infty$ target regions in the finite region game. As a consequence, we can compute all such states of

$\mathcal{R}(\mathcal{G})$ with complexity linear in the size of $\mathcal{R}(\mathcal{G})$, and remove them safely by Lemma 7.2. The same approach can be followed for the final weights $-\infty$.

Now, we can compute the set of configurations of value $+\infty$, with a similar technique. Notice that a configuration (ℓ, ν) cannot reach the target locations if and only if $(\ell, [\nu])$ is not in the attractor of Min to the targets in the finite region game. We can thus compute all such states of $\mathcal{R}(\mathcal{G})$ with complexity linear in the size of $\mathcal{R}(\mathcal{G})$, and remove them safely by Lemma 7.2.

We then decompose $\mathcal{R}(\mathcal{G})$ in SCCs. By Proposition 9.3, each SCC is either positive or negative (*i.e.* it contains only positive cycles, or only negative ones). Then, in order to find the sign of a component, it suffices to find one of its simple cycles, for example with a depth-first search, then compute the weight of one play following it.

As we did for weighted (untimed) games, we then compute values in inverse topological order over the SCCs. Once the values of all configurations in (ℓ, r) appearing in previously considered SCCs have been computed, they are no longer modified in further computation. This is the case, in particular, for all pairs (ℓ, r) that have value $+\infty$, that we precompute from the beginning. In order to resolve a positive SCC of $\mathcal{R}(\mathcal{G})$, we apply the value iteration operator $\mathcal{F}$ on the current piecewise affine function, only modifying the pieces appearing in the SCC, until reaching a fixpoint over these pieces. In order to resolve a negative SCC of $\mathcal{R}(\mathcal{G})$, we compute the attractor for Max to the previously computed SCCs: outside of this attractor, we set the value to $-\infty$. Then, we apply $\mathcal{F}$ for pieces appearing in the SCC, initialising them to $-\infty$ (equivalently, we compute in the dual game, that is a positive SCC), until reaching a fixpoint over these pieces. The next proposition contains the correction and termination arguments that where presented in Propositions 7.3, 7.4, and 7.5 for the untimed setting:

Proposition 10.2. *Let $\mathcal{G}$ be a divergent game with no configurations of value $+\infty$.*

1. *The value iteration algorithm applied on a positive SCC of $\mathcal{R}(\mathcal{G})$ with n region states stabilises after at most n steps.*

2. *In a negative SCC, region states (ℓ, r) of $\mathcal{R}(\mathcal{G})$ of value $-\infty$ are all the ones not in the attractor for Max to the targets.*

3. *The value iteration algorithm, initialised with $-\infty$, applied on a negative SCC of $\mathcal{R}(\mathcal{G})$ with n region states, and no configuration of value $-\infty$, stabilises after at most n steps.*

Proof of Proposition 10.2-1. There are no negative cycles in the SCC, therefore there are no configurations with value $-\infty$, and all values are finite. Let K be a bound on the values $|\boldsymbol{V}_\ell^n(\nu)|$ obtained after n steps of the algorithm.[6] Let us fix an integer $p > (2K + (n-1)^2 w_{\max})n$. We will show that the values obtained after $n + p$ steps are identical to those obtained after n steps only. Therefore, since the algorithm computes

[6] The value iteration emulates the attractor computation, so every value is finite after n steps. Moreover, functions $\nu \mapsto \boldsymbol{V}_\ell^n(\nu)$ are piecewise affine with a finite number of pieces over a bounded space, allowing us to obtain this uniform bound K.

non-increasing sequences of values, we have indeed stabilised after n steps only. Let us assume the existence of a configuration (ℓ, ν) such that $\boldsymbol{V}_\ell^{n+p}(\nu) < \boldsymbol{V}_\ell^n(\nu)$. By induction on p, we can show the existence of a configuration (ℓ', ν') and a finite play ρ from (ℓ, ν) to (ℓ', ν'), with length p and weight $\boldsymbol{V}_\ell^{n+p}(\nu) - \boldsymbol{V}_{\ell'}^n(\nu')$: the play is composed of the delays and transitions that optimise successively the min/max operator in $\mathcal{F}$.

Claim (timed setting notations)**.** *For all $i < j \in \mathbb{N}$, if $\boldsymbol{V}^j \neq \boldsymbol{V}^i$ then, for all configurations (ℓ, ν), there exists (ℓ', ν') and a play ρ from (ℓ, ν) to (ℓ', ν') with $|\rho| = j - i$ and $\mathsf{wt}_\Sigma(\rho) = \boldsymbol{V}_\ell^j(\nu) - \boldsymbol{V}_{\ell'}^i(\nu')$.*

Proof of Claim. Let us fix i, and prove it by induction on $j > i$.

Initialisation : If $j = i + 1$, we applied $\mathcal{F}$ once between $\boldsymbol{V}^i$ and $\boldsymbol{V}^j$, so for all configurations (ℓ, ν) there exists (ℓ', ν') and a transition $(\ell, \nu) \xrightarrow{d,t} (\ell', \nu')$ of weight $\boldsymbol{V}_\ell^j(\nu) - \boldsymbol{V}_{\ell'}^i(\nu')$.

Iteration : We assume the property holds for $j - 1 > i$, and $\boldsymbol{V}^j \neq \boldsymbol{V}^i$. We applied $\mathcal{F}$ once between $\boldsymbol{V}^{j-1}$ and $\boldsymbol{V}^j$, so for all configurations (ℓ, ν), there exists (ℓ', ν') and a transition $(\ell, \nu) \xrightarrow{d,t} (\ell', \nu')$ of weight $\boldsymbol{V}_\ell^j(\nu) - \boldsymbol{V}_{\ell'}^i(\nu')$. We apply the property on i and $j - 1$ ($\boldsymbol{V}^{j-1} \neq \boldsymbol{V}^i$ because $\boldsymbol{V}^j \neq \boldsymbol{V}^i$ and as soon as $\boldsymbol{V}$ stabilises, the fixpoint is reached and the iteration stops), and obtain that for all configurations (ℓ', ν'), there exists (ℓ'', ν'') and a play ρ from (ℓ', ν') to (ℓ'', ν'') with $|\rho| = j - 1 - i$ and $\mathsf{wt}_\Sigma(\rho) = \boldsymbol{V}_{\ell'}^{j-1}(\nu') - \boldsymbol{V}_{\ell''}^i(\nu'')$). Then we define $\rho' = (\ell, \nu) \xrightarrow{d,t} (\ell', \nu') \xrightarrow{\rho} (\ell'', \nu'')$ and it holds that $|\rho'| = j - i$ and $\mathsf{wt}_\Sigma(\rho') = \boldsymbol{V}_\ell^j(\nu) - \boldsymbol{V}_{\ell''}^i(\nu'')$. △ △

This finite play being of length greater than $(2K + (n-1)^2 w_{\max})n$, if we associate each visited configuration (ℓ, ν) to the region state $(\ell, [\nu])$, there is at least one state of $\mathcal{R}(\mathcal{G})$ appearing more than $2K + (n-1)^2 w_{\max}$ times. Thus, it can be decomposed into at least $2K + (n-1)^2 w_{\max}$ plays following cycles of $\mathcal{R}(\mathcal{G})$ and at most $(n-1)$ finite plays ρ_i' visiting each state of $\mathcal{R}(\mathcal{G})$ at most once. All cycles of the SCC being positive, the weight of ρ is at least $(2K + (n-1)^2 w_{\max}) - (n-1)^2 w_{\max} = 2K$, bounding from below each ρ_i''s weight by $-(n-1)w_{\max}$. Then, $\boldsymbol{V}_\ell^{n+p}(\nu) - \boldsymbol{V}_{\ell'}^n(\nu') \geqslant 2K$, so $\boldsymbol{V}_\ell^{n+p}(\nu) \geqslant 2K + \boldsymbol{V}_{\ell'}^n(\nu') \geqslant 2K - K \geqslant K$. But $K \geqslant \boldsymbol{V}_\ell^n(\nu)$, so $\boldsymbol{V}_\ell^{n+p}(\nu) \geqslant \boldsymbol{V}_\ell^n(\nu)$, and that is a contradiction. □

Much like in the untimed setting, negative SCCs can be resolved using a dual method. First, we characterise the $-\infty$ values as regions of $\mathcal{R}(\mathcal{G})$ where Max cannot unilaterally guarantee to reach the targets.

Proof of Proposition 10.2-2. Consider a state (ℓ, r) of $\mathcal{R}(\mathcal{G})$ in the attractor for Max to the targets. Then, if Max applies a winning memoryless strategy for the reachability objective to the target locations, for all $\nu \in r$, all strategies of Min will generate a play from (ℓ, ν) reaching a target after at most $|\mathcal{R}(\mathcal{G})|$ steps. This implies that (ℓ, ν) has a finite (lower) value in the game.

Reciprocally, if (ℓ, r) is not in the attractor, by determinacy of timed games with reachability objectives, for all $\nu \in r$, Min has a (memoryless) strategy σ_{Min} to ensure that no strategy of Max permits to reach a target location from (ℓ, ν). Applying σ_{Min} long

enough to generate a play following many negative cycles, before switching to a strategy allowing Min to reach the target (such a strategy exists since no configuration has value $+\infty$ in the game), allows Min to obtain from (ℓ, ν) a negative weight as small as possible. Thus, (ℓ, ν) has value $-\infty$. □

Thus, given a negative SCC, we can compute configurations of value $-\infty$ in time polynomial in the SCC's size. Then, finite values of other configurations can be computed by applying $\mathcal{F}$.

Proof of Proposition 10.2-3. From a negative SCC $\mathcal{G}$ that has no more configuration of value $+\infty$ or $-\infty$, consider the dual (positive) SCC $\widetilde{\mathcal{G}}$ obtained by: (i) switching locations of Min and Max; (ii) taking the opposite of every weight in locations and transitions. Sets of strategies of both players are exchanged in those two games, so that the upper value in $\mathcal{G}$ is equal to the opposite of the lower value in $\widetilde{\mathcal{G}}$, and vice versa. Since weighted games are determined, the value of $\mathcal{G}$ is the opposite of the value of $\widetilde{\mathcal{G}}$. Then, the value of $\mathcal{G}$ can be deduced from the value of $\widetilde{\mathcal{G}}$, for which Proposition 10.2-1 applies.

It is then immediate that the values computed with this computation of the smallest fixpoint of $\mathcal{F}$ are exactly the opposite values of the ones computed in the dual positive SCC. □

By Theorem 10.1, we obtain a triply-exponential algorithm computing the value of a divergent weighted timed game. This shows that the value problem is in 3-EXPTIME for divergent weighted timed game. The proof for EXPTIME-hardness comes from a reduction of the problem of solving timed games with reachability objectives [JT07]. To a reachability timed game, we simply add weights 1 on every edge and 0 on every location, making it a divergent weighted timed game. Then, Min wins the reachability timed game if and only if the value in the weighted timed game is lower than threshold $\alpha = |\mathcal{R}(\mathcal{G})|$. One direction of this statement's proof is direct by definition of having a value smaller than $+\infty$, and the other comes from the fact that reachability in the timed game implies reachability in the region game in less than α transitions, in turn implying that Min can ensure target reachability in the WTG with weight below α, *i.e.* $\mathsf{Val}_{\mathcal{G}}(s, \nu) \leqslant \alpha$.

In an SCC of $\mathcal{R}(\mathcal{G})$, the value iteration algorithm of [ABM04] allows us to compute an ε-optimal strategy for both players (for configurations having a finite value), that is constant (delay or fire an edge) over each piece of the piecewise affine value function. As in the untimed setting, we may then compose such ε-optimal strategies to obtain an ε'-optimal strategy in $\mathcal{G}$ (ε' is greater than ε, but can be controlled with respect to the number of SCCs in $\mathcal{R}(\mathcal{G})$).

11. Approximating values

In this chapter, we will study almost-divergent weighted timed games. Our goal is to prove Theorem 9.2: Given an almost-divergent WTG $\mathcal{G}$, a location ℓ and $\varepsilon \in \mathbb{Q}_{>0}$, we can compute an ε-approximation of $\mathsf{Val}_{\mathcal{G}}(\ell, \mathbf{0})$ in time triply-exponential in the size of $\mathcal{G}$ and polynomial in $1/\varepsilon$.

To obtain this result, we follow an approximation schema that we now outline. First, we will always reason on the region game $\mathcal{R}(\mathcal{G})$ of the almost-divergent WTG $\mathcal{G}$. The goal is to compute an ε-approximation of $\mathsf{Val}_{\mathcal{R}(\mathcal{G})}((\ell_0, [\mathbf{0}]), \mathbf{0})$ for some initial location ℓ_0. As already recalled, techniques of Chapter 10 allow one to compute the (exact) values of a WTG played on a finite tree, using the value iteration operator $\mathcal{F}$. The idea is thus to decompose as much as possible the game $\mathcal{R}(\mathcal{G})$ as a WTG over a tree. First, we decompose the region game into SCCs (left of Figure 11.1), and as in previous chapters we must think about the final weight functions as the previously computed approximations of the values of SCCs coming after the current one in the topological order. We will keep as an invariant that final weight functions are piecewise linear functions with a finite number of pieces, and are continuous on each region.

For an SCC of $\mathcal{R}(\mathcal{G})$ and an initial state $(\ell_0, [\mathbf{0}])$ of $\mathcal{R}(\mathcal{G})$ provided by the SCC decomposition, we follow Chapter 7.4 and show that the game on the SCC is equivalent to a game on a tree built from a semi-unfolding (see middle of Figure 11.1) of $\mathcal{R}(\mathcal{G})$ from $(\ell_0, [\mathbf{0}])$ of finite depth, with certain nodes of the tree being *kernels* (parts of $\mathcal{R}(\mathcal{G})$ that contain all cycles of weight 0). The semi-unfolding is stopped either when reaching a final location, or when some location (or kernel) has been visited for a certain fixed number of times.

Then, we compute an approximation of $\mathsf{Val}(\ell_0, \mathbf{0})$ with a bottom-up computation on the semi-unfolding. This computation is exact on nodes labelled by a single region state s, but approximated on kernel nodes K_s. For the latter, we use the corner-point abstraction (right of Figure 11.1) over $1/N$-regions to compute values, and prove that this provides an $1/N$-approximation of values.

Finally, we will use the semi-unfolding structure to derive our second result, stated in Theorem 9.3: it is a more symbolic approximation schema based on the value iteration algorithm only. It is more symbolic in the sense that it does not require the SCC decomposition, the computation of kernels nor the semi-unfolding of the game in a tree.

Remark. As in previous chapters, one can assume without loss of generality that final weights are finite. Then, it is easy to detect the set of states with value $+\infty$: these are all the states from which Min cannot ensure reachability of a target location $\ell \in L_{\mathsf{t}}$ with $\mathsf{wt}_{\mathsf{t}}(\ell) < +\infty$. It can therefore be computed by an attractor computation, and is indeed a property constant on each region. In particular, removing those states from $\mathcal{R}(\mathcal{G})$ does

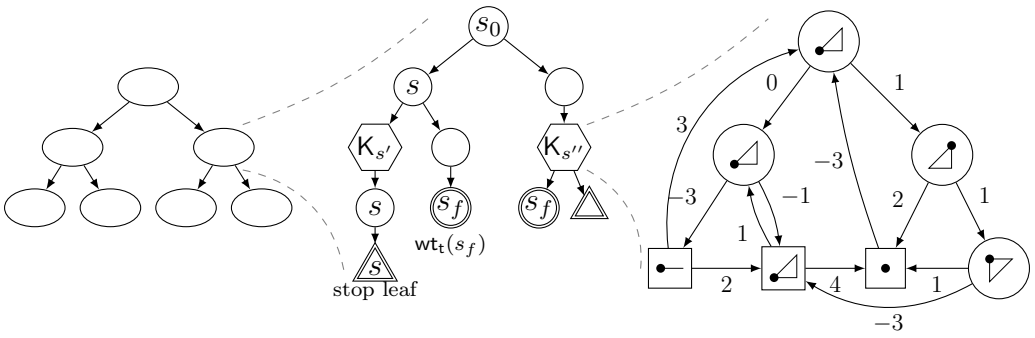


Figure 11.1.: Static approximation schema: SCC decomposition of $\mathcal{R}(\mathcal{G})$, semi-unfolding of an SCC, corner-point abstraction for the kernels

not affect the value of any other state and can be done in complexity linear in $|\mathcal{R}(\mathcal{G})|$. We will therefore assume that the considered WTG have no configurations with value $+\infty$, and no target configuration with final weight $+\infty$ or $-\infty$.

11.1. Kernel of an almost-divergent WTG

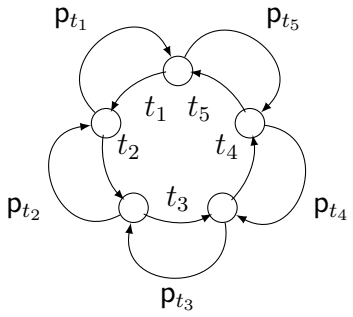
The approximation procedure described above uses *kernels* in order to group together all cycles of weight 0. We study those kernels and give a characterisation allowing computability. Contrary to the non-negative case studied in [BJM15], the situation is more complex in our arbitrary case, since weights of both locations and edges may differ from 0 in the kernel. Moreover, it is not trivial (and may not be true in a non almost-divergent WTG) to know whether it is sufficient to consider only simple cycles, *i.e.* cycles without repetitions.

We will now construct the kernel K as the subgraph of $\mathcal{R}(\mathcal{G})$ containing all 0-cycles. Formally, let T_K be the set of edges of $\mathcal{R}(\mathcal{G})$ belonging to a *simple* 0-cycle, and S_K be the set of states covered by T_K. We define the kernel K of $\mathcal{R}(\mathcal{G})$ as the subgraph of $\mathcal{R}(\mathcal{G})$ defined by S_K and T_K. Edges in $T \backslash T_\mathsf{K}$ with starting state in S_K are called the output edges of K. We define it using only simple 0-cycles in order to ensure its computability. However, we now show that this is of no harm, since the kernel contains exactly all the 0-cycles, which will be crucial in the approximation schema we present in Section 11.3.

Proposition 11.1. *A cycle of $\mathcal{R}(\mathcal{G})$ is entirely in K if and only if it is a 0-cycle.*

Proof. We prove that every 0-cycle is in K by induction on the length of the cycles. The initialisation contains only cycles of length 1, that are in K by construction. If we consider a cycle p of length $n > 1$, it is either simple or it can be rotated and decomposed into $\mathsf{p}'\mathsf{p}''$, p' and p'' being smaller cycles. Let ρ be a corner play following $\mathsf{p}'\mathsf{p}''$. We denote by ρ' the prefix of ρ following p' and ρ'' the suffix following p''. It holds that $\mathsf{wt}_\Sigma(\rho') = -\mathsf{wt}_\Sigma(\rho'')$, and in an almost-divergent SCC this implies $\mathsf{wt}_\Sigma(\rho') = \mathsf{wt}_\Sigma(\rho'') = 0$. Therefore, by Lemma 9.1 both p' and p'' are 0-cycles, and they must be in K by induction

hypothesis. Note that this reasoning proves that every cycle contained in a longer 0-cycle is also a 0-cycle.



We now prove that every cycle in K is a 0-cycle. By construction, every edge $t \in T_{\mathsf{K}}$ is part of a simple 0-cycle. Thus, to every edge $t \in T_{\mathsf{K}}$, we can associate a path p_t such that $t\mathsf{p}_t$ is a simple 0-cycle (rotate the simple cycle if necessary). We can prove (using both Lemmas 9.1 and 9.2) the following property, that was trivial in the untimed setting, by relying on another pumping argument on corners:

Claim. *If $t_1 \cdots t_n$ is a path in K, then $t_1 t_2 \cdots t_n \mathsf{p}_{t_n} \cdots \mathsf{p}_{t_2} \mathsf{p}_{t_1}$ is a 0-cycle of $\mathcal{R}(\mathcal{G})$.*

Proof of Claim. We prove the property by induction on n. For $n = 1$, the property is immediate since $t_1 \mathsf{p}_{t_1}$ is a 0-cycle. Consider then n such that the property holds for n, and let us prove that it holds for $n+1$. We will exhibit two corner plays following $t_1 \cdots t_{n+1} \mathsf{p}_{t_{n+1}} \cdots \mathsf{p}_{t_1}$ of opposite weight and conclude with Lemma 9.1.

Let v_0 be a corner of $\mathsf{last}(t_{n+1})$. Since $t_{n+1}\mathsf{p}_{t_{n+1}}$ is a 0-cycle, there exists $w \in \mathbb{Z}$, a corner play ρ_0 following t_{n+1} ending in v_0 with weight w and a corner play ρ'_0 following $\mathsf{p}_{t_{n+1}}$ beginning in v_0 with weight $-w$. We name v'_0 the corner of $\mathsf{last}(t_n)$ where ends ρ'_0. We consider any corner play ρ_1 following t_{n+1} from corner v'_0. The corner play $\rho'_0\rho_1$ follows the path $\mathsf{p}_{t_{n+1}} t_{n+1}$ that is also a 0-cycle by Lemma 9.2, therefore ρ_1 has weight w. We denote by v_1 the corner where ends ρ_1. By iterating this construction, we obtain some corner plays $\rho_0, \rho_1, \rho_2, \ldots$ following t_{n+1} and $\rho'_0, \rho'_1, \rho'_2, \ldots$ following $\mathsf{p}_{t_{n+1}}$ such that ρ'_i goes from corner v_i to v'_i, and ρ_{i+1} from corner v'_i to v_{i+1}, for all $i \geqslant 0$. Moreover, all corner plays ρ_i have weight w and all corner plays ρ'_i have weight $-w$. Consider the first index l such that $\mathsf{v}_l = \mathsf{v}_k$ for some $k < l$, which exists because the number of corners is finite.

We apply the induction to find a corner play following $t_1 \cdots t_n \mathsf{p}_{t_n} \cdots \mathsf{p}_{t_1}$, going through the corner v'_k in the middle: more formally, there exists w_α, a corner play ρ_α following $t_1 \cdots t_n$ ending in v'_k with weight w_α and a corner play ρ'_α following $\mathsf{p}_{t_n} \cdots \mathsf{p}_{t_1}$ beginning in v'_k with weight $-w_\alpha$. We apply the induction a second time with corner v'_{l-1}: there exists w_β, a corner play ρ_β following $t_1 \cdots t_n$ ending in v'_{l-1} with weight w_β and a corner play ρ'_β following $\mathsf{p}_{t_n} \cdots \mathsf{p}_{t_1}$ beginning in v'_{l-1} with weight $-w_\beta$.

The corner play $\rho_\alpha \rho_{k+1} \rho'_{k+1} \rho_{k+2} \rho'_{k+2} \cdots \rho'_{l-1} \rho'_\beta$, of weight $w_\alpha + (w-w)(l-k) - w_\beta = w_\alpha - w_\beta$, follows the cycle $t_1 \cdots t_n (t_{n+1}\mathsf{p}_{t_{n+1}})^{l-k} \mathsf{p}_{t_n} \cdots \mathsf{p}_{t_1}$. The corner play $\rho_\beta \rho_l \rho'_k \rho'_\alpha$, of weight $w_\beta + w - w - w_\alpha = w_\beta - w_\alpha$, follows the cycle $t_1 \cdots t_n t_{n+1} \mathsf{p}_{t_{n+1}} \mathsf{p}_{t_n} \cdots \mathsf{p}_{t_1}$. Since the game is almost-divergent, and those two corner plays are in the same SCC, both have weight 0. The second corner play of weight 0 ensures that the cycle $t_1 \cdots t_{n+1} \mathsf{p}_{t_{n+1}} \cdots \mathsf{p}_{t_1}$ is a 0-cycle, by Lemma 9.1. △

Now, if p is a cycle of $\mathcal{R}(\mathcal{G})$ in K, there exists a cycle p' such that pp' is a 0-cycle, therefore p is a 0-cycle. □

11.2. Semi-unfolding of almost-divergent WTGs

Given an almost-divergent WTG $\mathcal{G}$, we describe the construction of its *semi-unfolding* $\mathcal{T}(\mathcal{G})$ (as depicted in Figure 11.1). This crucially relies on the absence of states with value $-\infty$, so we explain how to deal with them first:

Lemma 11.1. *In an SCC of $\mathcal{R}(\mathcal{G})$, the set of configurations with value $-\infty$ is a union of regions computable in time linear in the size of $\mathcal{R}(\mathcal{G})$.*

Proof. If the SCC is non-negative, the cumulated weight cannot decrease along a cycle, thus, there can be no configuration with value $-\infty$.

If the SCC is non-positive, let T_{t} be the set of edges of $\mathcal{R}(\mathcal{G})$ whose end state has location in L_{t}. We can prove that a configuration has value $-\infty$ if and only if it belongs to a state where player Min can ensure the LTL formula on edges: $(\mathrm{G}\,\neg T_{\mathsf{t}}) \wedge \neg \mathrm{FG}\, T_{\mathsf{K}}$. The procedure to detect $-\infty$ states thus consists of four attractor computations, which can be done in time linear in $|\mathcal{R}(\mathcal{G})|$.

Let us prove that a configurations has value $-\infty$ if and only if it belongs to a state where player Min can ensure the LTL formula on edges: $\phi = (\mathrm{G}\,\neg T_{\mathsf{t}}) \wedge \neg \mathrm{FG}\, T_{\mathsf{K}}$. Since ω-regular games are determined, this is equivalent to saying that a configuration has finite value if and only if it belongs to a state where Max can ensure $\neg\phi$.

If (ℓ, r) is a region state where Min can ensure ϕ, he can ensure $-\infty$ value from all configurations in (ℓ, r) by avoiding S_{t} for as long as he desires, while not getting stuck in K, and thus going through an infinite number of negative cycles by Proposition 11.1. This proves that a state where Max cannot ensure $\neg\phi$ contains only valuations of value $-\infty$. Conversely, if (ℓ, r) is a state where Max can ensure $\neg\phi = (\mathrm{F}T_{\mathsf{t}}) \vee \mathrm{FG}T_{\mathsf{K}}$, then from (ℓ, r), Max must be able enforce either S_{t} reachability or staying in K forever. In both cases, Max can ensure a value above $-\infty$. □

We can now assume that no states of $\mathcal{G}$ have value $-\infty$, and that the final weight function maps all configurations to $\mathbb{R}$. Since wt_{t} is piecewise linear with finitely many pieces, wt_{t} is bounded. Let $w^{\mathsf{t}}_{\max}$ denote the supremum of $|\mathsf{wt}_{\mathsf{t}}|$, ranging over all target configurations.

We now explain how to build the semi-unfolding $\mathcal{T}(\mathcal{G})$. We only build the semi-unfolding $\mathcal{T}(\mathcal{G})$ of an SCC of $\mathcal{G}$ starting from some state $(\ell_0, r_0) \in S$ of the region game, since it is then easy to glue all the semi-unfoldings together to get the one of the full game. Since every configuration has finite value, we will prove that values of the game are bounded by $|\mathcal{R}(\mathcal{G})|w_{\max} + w^{\mathsf{t}}_{\max}$. As a consequence, we can find a bound γ linear in $|\mathcal{R}(\mathcal{G})|$, $w_{\max}$ and $w^{\mathsf{t}}_{\max}$ such that a play that visits some state outside the kernel more than γ times has weight strictly above $|\mathcal{R}(\mathcal{G})|w_{\max} + w^{\mathsf{t}}_{\max}$, hence is useless for the value computation. This leads to considering the semi-unfolding $\mathcal{T}(\mathcal{G})$ of $\mathcal{G}$ (nodes in the kernel are not unfolded, see Figure 11.1) such that each node not in the kernel is encountered

at most γ times along a branch: the end of each branch is called a *stopped leaf* of the semi-unfolding. In particular, the depth of $\mathcal{T}(\mathcal{G})$ is bounded by $|\mathcal{R}(\mathcal{G})|\gamma$, and thus is polynomial in $|\mathcal{R}(\mathcal{G})|$, $w_{\max}$ and $w^{\mathsf{t}}_{\max}$. Leaves of the semi-unfolding are thus of two types: target leaves that are copies of target locations of $\mathcal{G}$ for which we set the target weight as in $\mathcal{G}$, and stop leaves for which we set their target weight as being constant to $+\infty$ if the SCC $\mathcal{G}$ is non-negative, and $-\infty$ if the SCC is non-positive.

Proposition 11.2. *Let $\mathcal{G}$ be an almost-divergent WTG, and let (ℓ_0, r_0) be some state of the region game. We can define a semi-unfolding $\mathcal{T}(\mathcal{G})$ with initial state $(\tilde{\ell}_0, r_0)$ (a copy of state (ℓ_0, r_0)) which is equivalent to $\mathcal{G}$, i.e. for all $\nu_0 \in r_0$, $\mathsf{Val}_{\mathcal{G}}(\ell_0, \nu_0) = \mathsf{Val}_{\mathcal{T}(\mathcal{G})}((\tilde{\ell}_0, r_0), \nu_0)$.*

11.2.1. Semi-unfolding construction

In order to prove Proposition 11.2, we will construct the desired semi-unfolding $\mathcal{T}(\mathcal{G})$ of a (non-negative or non-positive) SCC $\mathcal{G}$, largely following the semi-unfolding of Chapter 7.4.

If (ℓ, r) is in K, we let $\mathsf{K}_{\ell,r}$ be the part of K accessible from (ℓ, r) (note that $\mathsf{K}_{\ell,r}$ is an SCC as K is a disjoint set of SCCs). We define the output edges of $\mathsf{K}_{\ell,r}$ as being the output edges of K accessible from (ℓ, r). If (ℓ, r) is not in K, the output edges of (ℓ, r) are the edges of $\mathcal{R}(\mathcal{G})$ starting in (ℓ, r).

Formally, we define a tree T whose nodes will either be labelled by region graph states $(\ell, r) \in S \backslash S_{\mathsf{K}}$ or by kernels $\mathsf{K}_{\ell,r}$, and whose edges will be labelled by output edges in $\mathcal{R}(\mathcal{G})$. The root of the tree T is labelled with (ℓ_0, r_0), or K_{ℓ_0,r_0} (if (ℓ_0, r_0) belongs to the kernel), and the successors of a node of T are then recursively defined by its output edges. When a state (ℓ, r) is reached by an output edge, the child is labelled by $\mathsf{K}_{\ell,r}$ if $(\ell, r) \in \mathsf{K}$, otherwise it is labelled by (ℓ, r). Edges in T are labelled by the edges used to create them. Along every branch, we stop the construction when either a final state is reached (*i.e.* a state not inside the current SCC) or the branch contains $3|\mathcal{R}(\mathcal{G})|w_{\max} + 2w^{\mathsf{t}}_{\max} + 2$ nodes labelled by the same state ((ℓ, r) or $\mathsf{K}_{\ell,r}$). Leaves of T with a location belonging to L_{t} are called *target leaves*, others are called *stopped leaves*.

We now transform T into a WTG $\mathcal{T}(\mathcal{G})$, by replacing every node labelled by a state (ℓ, r) by a different copy $(\tilde{\ell}, r)$ of (ℓ, r). Those states are said to inherit from (ℓ, r). Edges of T are replaced by the edges labelling them, and have a similar notion of inheritance. Every non-leaf node labelled by a kernel $\mathsf{K}_{\ell,r}$ is replaced by a copy of the WTG $\mathsf{K}_{\ell,r}$, output edges being plugged in the expected way. We deal with stopped leaves labelled by a kernel $\mathsf{K}_{\ell,r}$ by replacing them with a single node copy of (ℓ, r), like we dealt with node labelled by a state (ℓ, r). State partition between players and weights are inherited from the copied states of $\mathcal{R}(\mathcal{G})$. The only initial state of $\mathcal{T}(\mathcal{G})$ is the state denoted by $(\tilde{\ell}_0, r_0)$ inherited from (ℓ_0, r_0) in the root of T (either (ℓ_0, r_0) or K_{ℓ_0,r_0}). The final states of $\mathcal{T}(\mathcal{G})$ are the states derived from leaves of T. If $\mathcal{R}(\mathcal{G})$ is a non-negative (resp. non-positive) SCC, the final weight function wt_{t} is inherited from $\mathcal{R}(\mathcal{G})$ on target leaves and set to $+\infty$ (resp. $-\infty$) on stopped leaves.

11.2.2. Semi-unfolding correctness

We will now prove that Proposition 11.2 holds on this semi-unfolding $\mathcal{T}(\mathcal{G})$.

Lemma 11.2. *All finite plays in $\mathcal{R}(\mathcal{G})$ have cumulated weight (ignoring final weights) at least $-|\mathcal{R}(\mathcal{G})|w_{\max}$ in the non-negative case, and at most $|\mathcal{R}(\mathcal{G})|w_{\max}$ in the non-positive case. Moreover, values of the game are bounded by $|\mathcal{R}(\mathcal{G})|w_{\max} + w^{\mathsf{t}}_{\max}$.*

Proof. Suppose first that $\mathcal{R}(\mathcal{G})$ is a non-negative SCC. Consider a play ρ following a path p. This path p can be decomposed into $\mathsf{p} = \mathsf{p}_1\mathsf{p}_1^c \cdots \mathsf{p}_n\mathsf{p}_n^c$ such that every p_i^c is a cycle, and $\mathsf{p}_1 \dots \mathsf{p}_n$ is a simple path in $\mathcal{R}(\mathcal{G})$ (thus $\sum_{i=1}^{n} |\mathsf{p}_i| \leqslant |\mathcal{R}(\mathcal{G})|$). Let us define all plays ρ_i and ρ_i^c as the restrictions of ρ on p_i and p_i^c. Now, since all plays following cycles have cumulated weight at least 0,

$$\mathsf{wt}_\Sigma(\rho) = \sum_{i=1}^{n} \mathsf{wt}_\Sigma(\rho_i) + \mathsf{wt}_\Sigma(\rho_i^c) \geqslant \sum_{i=1}^{n} -w_{\max}|\rho_i| + 0 \geqslant -|\mathcal{R}(\mathcal{G})|w_{\max}\,.$$

Similarly, we can show that every play in a non-positive SCC has cumulated weight at most $|\mathcal{R}(\mathcal{G})|w_{\max}$.

For the bound on the values, consider again two cases. If $\mathcal{R}(\mathcal{G})$ is non-negative, consider any memoryless attractor strategy σ_{Min} for Min toward S_{t}. Since all states have values below $+\infty$, all plays obtained from strategies of Max will follow simple paths of $\mathcal{R}(\mathcal{G})$, that have cumulated weight at most $|\mathcal{R}(\mathcal{G})|w_{\max}$ in absolute value. Similarly, if $\mathcal{R}(\mathcal{G})$ is non-positive, following the proof of Lemma 11.1, since all values are above $-\infty$, Max can ensure $\neg\phi$, *i.e.* $(\mathrm{F}T_{\mathsf{t}}) \vee \mathrm{FG}T_{\mathsf{K}}$ on all states. Then we can construct a strategy σ_{Max} for Max combining an attractor strategy toward S_{t} on states satisfying $\mathrm{F}T_{\mathsf{t}}$, a safety strategy on states satisfying GT_{K}, and an attractor strategy toward the latter on all other states. Then, all plays obtained from strategies of Min will either not be winning (GT_{K}) or follow simple paths of $\mathcal{R}(\mathcal{G})$. Both cases imply that the values of the game are bounded by $|\mathcal{R}(\mathcal{G})|w_{\max} + w^{\mathsf{t}}_{\max}$. □

Lemma 11.3. *All plays in $\mathcal{T}(\mathcal{G})$ from the initial state to a stopped leaf have cumulated weight at least $2|\mathcal{R}(\mathcal{G})|w_{\max} + 2w^{\mathsf{t}}_{\max} + 1$ if the SCC $\mathcal{R}(\mathcal{G})$ is non-negative, and at most $-2|\mathcal{R}(\mathcal{G})|w_{\max} - 2w^{\mathsf{t}}_{\max} - 1$ if it is non-positive.*

Proof. Note that by construction, all finite paths in $\mathcal{T}(\mathcal{G})$ from the initial state to a stopped leaf can be decomposed as $\mathsf{p}'\mathsf{p}_1 \cdots \mathsf{p}_{3|\mathcal{R}(\mathcal{G})|w_{\max}+2w^{\mathsf{t}}_{\max}+1}$ with all p_i being cycles around the same state. Additionally, those cycles cannot be 0-cycles by Proposition 11.1, since they take at least one edge outside of K. Therefore the restriction of ρ to $\mathsf{p}_1 \cdots \mathsf{p}_{3|\mathcal{R}(\mathcal{G})|w_{\max}+2w^{\mathsf{t}}_{\max}+1}$ has weight at least $3|\mathcal{R}(\mathcal{G})|w_{\max} + 2w^{\mathsf{t}}_{\max} + 1$ (in the non-negative case) and at most $-3|\mathcal{R}(\mathcal{G})|w_{\max} - 2w^{\mathsf{t}}_{\max} - 1$ (in the non-positive case). The beginning of the play, following p', has cumulated weight at least $-|\mathcal{R}(\mathcal{G})|w_{\max}$ (in the non-negative case) and at most $|\mathcal{R}(\mathcal{G})|w_{\max}$ (in the non-positive case), by Lemma 11.2. □

Consider two plays in $\mathcal{R}(\mathcal{G})$ and $\mathcal{T}(\mathcal{G})$, respectively:

$$\rho = ((\ell_1, r_1), \nu_1) \xrightarrow{d_1,t_1} \cdots \xrightarrow{d_{n-1},t_{n-1}} ((\ell_n, r_n), \nu_n)$$

$$\tilde{\rho} = ((\tilde{\ell}_1, r_1), \nu_1) \xrightarrow{d_1,\tilde{t}_1} \cdots \xrightarrow{d_{n-1},\tilde{t}_{n-1}} ((\tilde{\ell}_n, r_n), \nu_n)\,.$$

They are said to *mimic* each other if every $(\tilde{\ell}_i, r_i)$ is inherited from (ℓ_i, r_i) and every edge $\tilde{t}_i$ is inherited from the edge e_i. Combining Lemmas 11.3 and 11.2, we obtain:

Lemma 11.4. *If $\mathcal{R}(\mathcal{G})$ is a non-negative (resp. non-positive) SCC, every play from the initial state and with cumulated weight less than $|\mathcal{R}(\mathcal{G})|w_{\max} + 2w^{\mathsf{t}}_{\max} + 1$ (resp. greater than $-|\mathcal{R}(\mathcal{G})|w_{\max} - 2w^{\mathsf{t}}_{\max} - 1$) can be mimicked in $\mathcal{T}(\mathcal{G})$ without reaching a stopped leaf. Conversely, every play in $\mathcal{T}(\mathcal{G})$ reaching a target leaf can be mimicked in $\mathcal{R}(\mathcal{G})$.*

Proof. We prove only the non-negative case. Let ρ be a play of $\mathcal{R}(\mathcal{G})$ with cumulated weight less than $|\mathcal{R}(\mathcal{G})|w_{\max} + 2w^{\mathsf{t}}_{\max} + 1$. Consider the branch of the unfolded game it follows. If ρ cannot be mimicked in $\mathcal{T}(\mathcal{G})$, then a prefix of ρ reaches the stopped leaf of that branch when mimicked in $\mathcal{T}(\mathcal{G})$. In this situation, ρ starts by a prefix of weight at least $2|\mathcal{R}(\mathcal{G})|w_{\max} + 2w^{\mathsf{t}}_{\max} + 1$ by Lemma 11.3 and then ends with a suffix play of weight at least $-|\mathcal{R}(\mathcal{G})|w_{\max}$ by Lemma 11.2, and that contradicts the initial assumption. The non-positive case is proved exactly the same way, and the converse is true by construction. □

Then, the plays of $\mathcal{R}(\mathcal{G})$ starting in an initial configuration that cannot be mimicked in $\mathcal{T}(\mathcal{G})$ are not useful for value computation, which is formalised by Proposition 11.3:

Proposition 11.3. *For all valuations $\nu_0 \in r_0$, $\mathsf{Val}_{\mathcal{G}}(\ell_0, \nu_0) = \mathsf{Val}_{\mathcal{T}(\mathcal{G})}((\tilde{\ell}_0, r_0), \nu_0)$.*

Proof. By Lemma 8.1, we already know that $\mathsf{Val}_{\mathcal{G}}(\ell_0, \nu_0) = \mathsf{Val}_{\mathcal{R}(\mathcal{G})}((\ell_0, r_0), \nu_0)$. Recall that we only left finite values in $\mathcal{R}(\mathcal{G})$ (in the final weight functions, in particular), and more precisely $|\mathsf{Val}_{\mathcal{R}(\mathcal{G})}((\ell_0, r_0), \nu_0)| \leqslant |\mathcal{R}(\mathcal{G})|w_{\max} + w^{\mathsf{t}}_{\max}$ by Lemma 11.2. We first show that the value is also finite in $\mathcal{T}(\mathcal{G})$. Indeed, if $\mathsf{Val}_{\mathcal{T}(\mathcal{G})}((\tilde{\ell}_0, r_0), \nu_0) = +\infty$, since we assumed all final weights of $\mathcal{R}(\mathcal{G})$ bounded, we are necessarily in the non-negative case, and Max is able to ensure stopped leaves reachability.

Claim. *If $\mathsf{Val}_{\mathcal{T}(\mathcal{G})}((\tilde{\ell}_0, r_0), \nu_0) = +\infty$, then there are no winning strategies in $\mathcal{R}(\mathcal{G})$ for Min ensuring weight less than $|\mathcal{R}(\mathcal{G})|w_{\max} + w^{\mathsf{t}}_{\max} + 1$ from (ℓ_0, r_0).*

Thus, we can obtain the contradiction $\mathsf{Val}_{\mathcal{R}(\mathcal{G})}((\ell_0, r_0), \nu_0) > |\mathcal{R}(\mathcal{G})|w_{\max} + w^{\mathsf{t}}_{\max}$.

Proof of Claim. By contradiction, consider a strategy σ_{Min} of Min ensuring weight $A \leqslant |\mathcal{R}(\mathcal{G})|w_{\max} + w^{\mathsf{t}}_{\max} + 1$ in $\mathcal{R}(\mathcal{G})$. Then, for all σ_{Max}, the cumulated weight of $\mathsf{play}_{\mathcal{R}(\mathcal{G})}(((\ell_0, r_0), \nu_0), \sigma_{\mathsf{Min}}, \sigma_{\mathsf{Max}})$ (reaching target configuration (ℓ, ν)) is at most $A - \mathsf{wt}_{\mathsf{t}}(\ell, \nu) \leqslant |\mathcal{R}(\mathcal{G})|w_{\max} + 2w^{\mathsf{t}}_{\max} + 1$, and by Lemma 11.4 this play does not reach a stopped leaf when mimicked in $\mathcal{T}(\mathcal{G})$, which is absurd. △

If $\mathsf{Val}_{\mathcal{T}(\mathcal{G})}((\tilde{\ell}_0, r_0), \nu_0) = -\infty$, we are necessarily in the non-positive case, and by construction this implies having Min ensuring stopped leaves reachability in $\mathcal{T}(\mathcal{G})$.

Claim. *If $\mathsf{Val}_{\mathcal{T}(\mathcal{G})}((\tilde{\ell}_0, r_0), \nu_0) = -\infty$, then there are no winning strategies in $\mathcal{R}(\mathcal{G})$ for Max ensuring weight above $-|\mathcal{R}(\mathcal{G})|w_{\max} - w^{\mathsf{t}}_{\max} - 1$ from (ℓ_0, r_0).*

Thus, we can obtain the contradiction $\mathsf{Val}_{\mathcal{R}(\mathcal{G})}((\ell_0, r_0), \nu_0) < -|\mathcal{R}(\mathcal{G})|w_{\max} - w^{\mathsf{t}}_{\max}$.

Proof of Claim. By contradiction, consider a strategy σ_{Max} of Max ensuring weight $A \geqslant -|\mathcal{R}(\mathcal{G})|w_{\max} - w^{\mathsf{t}}_{\max} - 1$ in $\mathcal{R}(\mathcal{G})$. Then, for all σ_{Min}, the cumulated weight of $\mathsf{play}_{\mathcal{R}(\mathcal{G})}(((\tilde{\ell}_0, r_0), \nu_0), \sigma_{\mathsf{Min}}, \sigma_{\mathsf{Max}})$ (reaching target configuration (ℓ, ν)) is at least $A - \mathsf{wt}_{\mathsf{t}}(\ell, \nu) \geqslant -|\mathcal{R}(\mathcal{G})|w_{\max} - 2w^{\mathsf{t}}_{\max} - 1$, and by Lemma 11.4 this play does not reach a stopped leaf when mimicked in $\mathcal{T}(\mathcal{G})$, which is absurd. △

If $\mathcal{R}(\mathcal{G})$ is non-negative, for all $\varepsilon > 0$ we can fix an ε-optimal strategy σ_{Min} for Min in $\mathcal{T}(\mathcal{G})$. It is a winning strategy, so every play derived from σ_{Min} in $\mathcal{T}(\mathcal{G})$ reaches a target leaf, and can be mimicked in $\mathcal{R}(\mathcal{G})$ by Lemma 11.4. Therefore, σ_{Min} can be mimicked in $\mathcal{R}(\mathcal{G})$, where it is also winning, with the same value. From this we deduce $\mathsf{Val}_{\mathcal{R}(\mathcal{G})}((\ell_0, r_0), \nu_0) \leqslant \mathsf{Val}_{\mathcal{T}(\mathcal{G})}((\tilde{\ell}_0, r_0), \nu_0)$. If $\mathcal{R}(\mathcal{G})$ is non-positive, the same reasoning applies by considering an ε-optimal strategy for Max in $\mathcal{T}(\mathcal{G})$.

Let us now show the reverse inequality. If $\mathcal{R}(\mathcal{G})$ is non-negative, let us fix $0 < \varepsilon < 1$, an ε-optimal strategy σ_{Min} for Min in $\mathcal{R}(\mathcal{G})$, and a strategy σ_{Max} of Max in $\mathcal{R}(\mathcal{G})$. Let ρ be their outcome $\mathsf{play}_{\mathcal{R}(\mathcal{G})}(((\ell_0, r_0), \nu_0), \sigma_{\mathsf{Min}}, \sigma_{\mathsf{Max}})$, ρ_k be the finite prefix of ρ defining its cumulative weight and (ℓ_k, ν_k) be the configuration defining its final weight, such that $\mathsf{wt}_{\mathcal{R}(\mathcal{G})}(\rho) = \mathsf{wt}_{\Sigma}(\rho_k) + \mathsf{wt}_{\mathsf{t}}(\ell_k, \nu_k)$. Then,

$$\mathsf{wt}_{\mathcal{R}(\mathcal{G})}(\rho) \leqslant \mathsf{Val}_{\mathcal{R}(\mathcal{G})}((\ell_0, r_0), \nu_0) + \varepsilon < |\mathcal{R}(\mathcal{G})|w_{\max} + w^{\mathsf{t}}_{\max} + 1\,,$$

therefore

$$\mathsf{wt}_{\Sigma}(\rho_k) < |\mathcal{R}(\mathcal{G})|w_{\max} + w^{\mathsf{t}}_{\max} + 1 - \mathsf{wt}_{\mathsf{t}}(\ell_k, \nu_k) \leqslant |\mathcal{R}(\mathcal{G})|w_{\max} + 2w^{\mathsf{t}}_{\max} + 1\,,$$

and by Lemma 11.4 all such plays ρ can be mimicked in $\mathcal{T}(\mathcal{G})$, so that

$$\mathsf{Val}_{\mathcal{T}(\mathcal{G})}((\tilde{\ell}_0, r_0), \nu_0) \leqslant \mathsf{Val}_{\mathcal{R}(\mathcal{G})}((\ell_0, r_0), \nu_0)\,.$$

Once again, if $\mathcal{R}(\mathcal{G})$ is non-positive, the same reasoning applies by considering an ε-optimal strategy for Max in $\mathcal{R}(\mathcal{G})$. □

This proof not only holds on an SCC, but also on full almost-divergent WTGs, by simply stacking the semi-unfoldings of each SCC on top of each others.

Note that the semi-unfolding procedure of an SCC depends on $w^{\mathsf{t}}_{\max}$, where wt_{t} can be the value function of an SCCs under the current one. Assuming all configurations have finite value, we can extend the reasoning of Lemma 11.2 and bound all values in the full game by $|\mathcal{R}(\mathcal{G})|w_{\max} + w^{\mathsf{t}}_{\max}$, which lets us bound uniformly the unfolding depth of each SCC and gives us a bound on the depth of the complete semi-unfolding tree: $|\mathcal{R}(\mathcal{G})|(5|\mathcal{R}(\mathcal{G})|w_{\max} + 2w^{\mathsf{t}}_{\max} + 2) + 1$.

11.3. Approximation of almost-divergent WTGs

11.3.1. Approximation of kernels

We start by approximating a kernel $\mathcal{G}$ by extending the region-based approximation schema of [BJM15]. In their setting, all runs in kernels had weight 0, allowing a reduction to a finite weighted game. In our setting, we have to approximate the timed dynamics of runs, and therefore resort to the corner-point abstraction (as shown to the right of Figure 11.1).

Since final weight functions are piecewise linear with a finite number of pieces and continuous on regions, they are Λ-Lipschitz-continuous, for a given constant $\Lambda \geqslant 0$. We let $\mathbf{B} = w^L_{\max} \, |L||\mathsf{Reg}(\mathcal{X}, M)| + \Lambda$.

Let N be an integer. Consider the corner-point abstraction game $\Gamma_N(\mathcal{G})$ described in Chapter 8, with locations of the form (ℓ, r, v) with v a corner of the $1/N$-region r. Two plays ρ of $\mathcal{G}$ and ρ' of $\Gamma_N(\mathcal{G})$ are said to be $1/N$-*close* if they follow the same path p in $\mathcal{R}_N(\mathcal{G})$. In particular, at each step the configurations (ℓ, ν) in ρ and (ℓ', r', v') in ρ' (with v' a corner of the $1/N$-region v') satisfy $\ell = \ell'$ and $\nu \in r'$, and the edges taken in both plays have the same discrete weights. Close plays have *close* weights, in the following sense:

Lemma 11.5. *For all $1/N$-close plays ρ of $\mathcal{G}$ and ρ' of $\Gamma_N(\mathcal{G})$,*

$$|\mathsf{wt}_{\mathcal{G}}(\rho) - \mathsf{wt}_{\Gamma_N(\mathcal{G})}(\rho')| \leqslant \mathbf{B}/N \, .$$

Proof. Since ρ and ρ' follow the same locations ℓ of $\mathcal{G}$, one reaches a target location if and only if the other does. In the case where they do not reach a target location, both weights are infinite, and thus equal. We now look at the case where both plays reach a target location, moreover in the same step.

Consider the region path p of the run ρ: p can be decomposed into a simple path with maximal cycles in it. The number of such maximal cycles is bounded by $|L \times \mathsf{Reg}(\mathcal{X}, M)|$ and the remaining simple path has length at most $|L \times \mathsf{Reg}(\mathcal{X}, M)|$. Since all cycles of a kernel are 0-cycles, the parts of ρ that follow the maximal cycles have weight exactly 0.

Consider the same decomposition for the play ρ'. Cycles of p do not necessarily map to cycles over locations of $\Gamma_N(\mathcal{G})$, since the $1/N$-regions could be distinct. However, Lemma 8.3 shows that, for all those cycles of p, there exists a sequence of finite plays of $\mathcal{G}$ whose weight tends to the weight of ρ'. Since all those finite plays follow a cycle of the region game $\mathcal{R}(\mathcal{G})$ (with $\mathcal{G}$ being a kernel), they all have weight 0. Hence, the parts of ρ' that follow the maximal cycles of p have also weight exactly 0.

Therefore, the difference $|\mathsf{wt}_{\mathcal{G}}(\rho) - \mathsf{wt}_{\Gamma_N(\mathcal{G})}(\rho')|$ is concentrated on the remaining simple path of p: on each edge of this path, the maximal weight difference is $1/N \times w^L_{\max}$ since $1/N$ is the largest difference possible in time delays between plays that stay $1/N$-close (since they stay in the same $1/N$-regions). Moreover, the difference between the final weight functions is bounded by Λ/N, since the final weight function wt_{t} is Λ-Lipschitz-continuous and the final weight function of $\Gamma_N(\mathcal{G})$ is obtained as limit of wt_{t}. Summing the two contributions, we obtain as upper bound the constant $\mathbf{B}/N$. □

In particular, if we start in configurations (ℓ_0, ν_0) of $\mathcal{G}$, and $((\ell_0, r_0, \mathsf{v}_0), \mathsf{v}_0)$ of $\Gamma_N(\mathcal{G})$, with $\nu_0 \in r_0$, since both players have the ability to stay $1/N$-close all along the plays, a bisimulation argument permits to obtain that the values of the two games are also close in (ℓ_0, ν_0) and $((\ell_0, r_0, \mathsf{v}_0), \mathsf{v}_0)$:

Lemma 11.6. *For all locations $\ell \in L$, $1/N$-regions r, $\nu \in r$ and corners v of r, $|\mathsf{Val}_{\mathcal{G}}(\ell, \nu) - \mathsf{Val}_{\Gamma_N(\mathcal{G})}((\ell, r, \mathsf{v}), \mathsf{v})| \leqslant \mathbf{B}/N$.*

Proof. Let us prove that for $\alpha = \mathbf{B}/N$,

$$\mathsf{Val}_{\mathcal{G}}(\ell, \nu) \leqslant \mathsf{Val}_{\Gamma_N(\mathcal{G})}((\ell, r, \mathsf{v}), \mathsf{v}) + \alpha\,, \text{ and}$$

$$\mathsf{Val}_{\Gamma_N(\mathcal{G})}((\ell, r, \mathsf{v}), \mathsf{v}) \leqslant \mathsf{Val}_{\mathcal{G}}(\ell, \nu) + \alpha\,.$$

By definition and determinacy of turn-based WTG, this is equivalent to proving these two inequalities:

$$\inf_{\sigma_{\mathsf{Min}}} \sup_{\sigma_{\mathsf{Max}}} \mathsf{wt}_{\mathcal{G}}(\mathsf{play}((\ell, \nu), \sigma_{\mathsf{Min}}, \sigma_{\mathsf{Max}})) \leqslant \inf_{\sigma'_{\mathsf{Min}}} \sup_{\sigma'_{\mathsf{Max}}} \mathsf{wt}_{\Gamma_N(\mathcal{G})}(\mathsf{play}(((\ell, r, \mathsf{v}), \mathsf{v}), \sigma'_{\mathsf{Min}}, \sigma'_{\mathsf{Max}})) + \alpha$$

$$\sup_{\sigma'_{\mathsf{Max}}} \inf_{\sigma'_{\mathsf{Min}}} \mathsf{wt}_{\Gamma_N(\mathcal{G})}(\mathsf{play}(((\ell, r, \mathsf{v}), \mathsf{v}), \sigma'_{\mathsf{Min}}, \sigma'_{\mathsf{Max}})) \leqslant \sup_{\sigma_{\mathsf{Max}}} \inf_{\sigma_{\mathsf{Min}}} \mathsf{wt}_{\mathcal{G}}(\mathsf{play}((\ell, \nu), \sigma_{\mathsf{Min}}, \sigma_{\mathsf{Max}})) + \alpha$$

Consider the following equation:

$$|\mathsf{wt}_{\mathcal{G}}(\mathsf{play}((\ell, \nu), \sigma_{\mathsf{Min}}, \sigma_{\mathsf{Max}})) - \mathsf{wt}_{\Gamma_N(\mathcal{G})}(\mathsf{play}(((\ell, r, \mathsf{v}), \mathsf{v}), \sigma'_{\mathsf{Min}}, \sigma'_{\mathsf{Max}}))| \leqslant \alpha\,. \tag{11.1}$$

To show the first inequality, it suffices to show that for all σ'_{Min}, there exists σ_{Min} such that for all σ_{Max}, there is σ'_{Max} verifying (11.1). For the second, it suffices to show that for all σ'_{Max}, there exists σ_{Max} such that for all σ_{Min}, there is σ'_{Min} verifying (11.1). We will detail the proof for the first, the second being syntactically the same, with both players swapped.

Equation (11.1) can be obtained from Lemma 11.5, under the condition that the plays $\mathsf{play}((\ell, \nu), \sigma_{\mathsf{Min}}, \sigma_{\mathsf{Max}})$ and $\mathsf{play}(((\ell, r, \mathsf{v}), \mathsf{v}), \sigma'_{\mathsf{Min}}, \sigma'_{\mathsf{Max}})$ are $1/N$-close. Therefore, we fix a strategy σ'_{Min} of Min in the game $\Gamma_N(\mathcal{G})$, and we construct a strategy σ_{Min} of Min in $\mathcal{G}$, as well as two mappings $f\colon \mathsf{FPlays}^{\mathsf{Min}}_{\mathcal{G}} \to \mathsf{FPlays}^{\mathsf{Min}}_{\Gamma_N(\mathcal{G})}$ and $g\colon \mathsf{FPlays}^{\mathsf{Max}}_{\Gamma_N(\mathcal{G})} \to \mathsf{FPlays}^{\mathsf{Max}}_{\mathcal{G}}$ such that:

- for all $\rho \in \mathsf{FPlays}^{\mathsf{Min}}_{\mathcal{G}}$, ρ and $f(\rho)$ are $1/N$-close, and if ρ is consistent with σ_{Min} and starts in (ℓ, ν), then $f(\rho)$ is consistent with σ'_{Min} and starts in $((\ell, r, \mathsf{v}), \mathsf{v})$;
- for all $\rho' \in \mathsf{FPlays}^{\mathsf{Max}}_{\Gamma_N(\mathcal{G})}$, $g(\rho')$ and ρ' are $1/N$-close, and if ρ' is consistent with σ'_{Min} and starts in $((\ell, r, \mathsf{v}), \mathsf{v})$, then $g(\rho')$ is consistent with σ_{Min} and starts in (ℓ, ν).

We build σ_{Min}, f, and g by induction on the length n of plays, over prefixes of plays of length $n-1$, n and n, respectively. For $n = 0$ (plays of length 0 are those restricted to a single configuration), we let $f(\ell, \nu) = ((\ell, r, \mathsf{v}), \mathsf{v})$ and $g((\ell, r, \mathsf{v}), \mathsf{v}) = (\ell, \nu)$, leaving the other values arbitrary (since we will not use them).

Then, we suppose σ_{Min}, f, and g built until length $n-1$, n and n, respectively (if $n=0$, σ_{Min} has not been build yet), and we define them on plays of length n, $n+1$ and $n+1$, respectively. For every $\rho \in \mathsf{FPlays}_{\mathcal{G}}^{\mathsf{Min}}$ of length n, we note $\rho' = f(\rho)$. Consider the decision $(d', e') = \sigma'_{\mathsf{Min}}(\rho')$ and ρ'_+ the prefix ρ' extended with the decision (d', e'). By timed bisimulation, there exists (d, e) such that the prefix ρ_+ composed of ρ extended with the decision (d, e) builds $1/N$-close plays ρ_+ and ρ'_+. We let $\sigma_{\mathsf{Min}}(\rho) = (d, e)$. If $\rho_+ \in \mathsf{FPlays}_{\mathcal{G}}^{\mathsf{Min}}$, we also let $f(\rho_+) = \rho'_+$, and otherwise we let $g(\rho'_+) = \rho_+$. Symmetrically, consider $\rho' \in \mathsf{FPlays}_{\Gamma_N(\mathcal{G})}^{\mathsf{Max}}$ of length n, and $\rho = g(\rho')$. For all possible decisions (d', e'), by timed bisimulation, there exists a decision (d, e) in the prefix ρ such that the respective extended plays ρ'_+ and ρ_+ are $1/N$-close. We then let $g(\rho'_+) = \rho_+$ if $\rho_+ \in \mathsf{FPlays}_{\mathcal{G}}^{\mathsf{Max}}$ and $f(\rho_+) = \rho'_+$ otherwise. We extend the definition of f and g arbitrarily for other prefixes of plays. The properties above are then trivially verified.

We then fix a strategy σ_{Max} of Max in the game $\mathcal{G}$, which determines a unique play $\mathsf{play}((\ell, \nu), \sigma_{\mathsf{Min}}, \sigma_{\mathsf{Max}})$. We construct a strategy σ'_{Max} of Max in the game $\Gamma_N(\mathcal{G})$ by building the unique play $\mathsf{play}(((\ell, r, \mathsf{v}), \mathsf{v}), \sigma'_{\mathsf{Min}}, \sigma'_{\mathsf{Max}})$ we will be interested in, such that each of its prefixes is in relation, via f or g, to the associated prefix of $\mathsf{play}((\ell, \nu), \sigma_{\mathsf{Min}}, \sigma_{\mathsf{Max}})$. Thus, we only need to consider a prefix of play $\rho' \in \mathsf{FPlays}_{\Gamma_N(\mathcal{G})}^{\mathsf{Max}}$ that starts in $((\ell, r, \mathsf{v}), \mathsf{v})$ and is consistent with σ'_{Min}, and σ'_{Max} built so far. Consider the play $\rho = g(\rho')$, starting in (ℓ, ν) and consistent with σ_{Min}, and σ_{Max} (by assumption). For the decision $(d, e) = \sigma_{\mathsf{Max}}(\rho)$ (letting ρ_+ be the extended prefix), the definition of f and g ensures that there exists a decision (d', e') after ρ' that results in an extended play ρ'_+ that is $1/N$-close, via f or g, with ρ_+. We thus can choose $\sigma'_{\mathsf{Max}}(\rho') = (d', e')$.

We finally have built two plays $\mathsf{play}((\ell, \nu), \sigma_{\mathsf{Min}}, \sigma_{\mathsf{Max}})$ and $\mathsf{play}((\ell', \nu'), \sigma'_{\mathsf{Min}}, \sigma'_{\mathsf{Max}})$ that are $1/N$-close, as needed, which concludes this proof. □

Using this result, picking N an integer larger than $\mathbf{B}/\varepsilon$, we can thus obtain $|\mathsf{Val}_{\mathcal{G}}(\ell, \nu) - \mathsf{Val}_{\Gamma_N(\mathcal{G})}((\ell, r, \mathsf{v}), \mathsf{v})| \leqslant \varepsilon$. Recall that $\Gamma_N(\mathcal{G})$ can be considered as an untimed weighted game (with reachability objective). Thus we can apply the result of [BGHM16], where it is shown that the optimal values of such games can be computed in pseudo-polynomial time (*i.e.* polynomial time with weights encoded in unary, instead of binary). We then define an ε-approximation of $\mathsf{Val}_{\mathcal{G}}$, named Val'_N, on each $1/N$-region by interpolating the values of its $1/N$-corners in $\Gamma_N(\mathcal{G})$ with a piecewise linear function: therefore, we can control the Lipschitz constant of the approximated value for further use.

Lemma 11.7. *Val'_N is an ε-approximation of $\mathsf{Val}_{\mathcal{G}}$, that is piecewise linear with a finite number of pieces and $2\mathbf{B}$-Lipschitz-continuous over regions.*

Proof. By construction, the approximated value is piecewise linear with one piece per $1/N$-region. To prove the Lipschitz constant, it is then sufficient to bound the difference between $\mathsf{Val}_{\Gamma_N(\mathcal{G})}((\ell, r, \mathsf{v}), \mathsf{v})$ and $\mathsf{Val}_{\Gamma_N(\mathcal{G})}((\ell, r, \mathsf{v}'), \mathsf{v}')$, for v and v' two corners of a $1/N$-region r. We can pick any valuation ν in r and apply Lemma 11.6 twice, between ν and v, and between ν and v'. We obtain $|\mathsf{Val}_{\Gamma_N(\mathcal{G})}((\ell, r, \mathsf{v}), \mathsf{v}) - \mathsf{Val}_{\Gamma_N(\mathcal{G})}((\ell, r, \mathsf{v}'), \mathsf{v}')| \leqslant 2\mathbf{B}/N = 2\|\mathsf{v} - \mathsf{v}'\|_\infty \mathbf{B}$. □

11.3.2. Approximation of almost-divergent WTGs

We now explain how to approximate the value of an almost-divergent WTG $\mathcal{G}$, thus proving Theorem 9.2. First, we compute a semi-unfolding $\mathcal{T}(\mathcal{G})$ as described in the previous section. Then we perform a bottom-up computation of the approximation. As already recalled, techniques of Chapter 10 allow us to compute exact values of a tree-shape WTG. In consequence, we know how to compute the value of a non-kernel node of $\mathcal{T}(\mathcal{G})$, depending of the values of its children. There is no approximation needed here, so that if all children are ε-approximation, we can compute an ε-approximation of the node. Therefore, the only approximation lies in the kernels, and we explained before how to compute an arbitrarily close approximation of the value of a kernel. We crucially rely on the fact that the value function is 1-Lipschitz-continuous[1]. This entails that imprecisions will sum up along the bottom-up computations, as computing an ε-approximation of the value of a game whose final weights are ε'-approximations yields an $(\varepsilon+\varepsilon')$-approximation. Therefore we compute approximations with threshold $\varepsilon' = \varepsilon/\alpha$ for kernels in $\mathcal{T}(\mathcal{G})$, where α is the maximal number of kernels along a branch of $\mathcal{T}(\mathcal{G})$: α is smaller than the depth of $\mathcal{T}(\mathcal{G})$, which is bounded by Proposition 11.2.

The subregion granularity considered before for kernel approximation crucially depends on the Lipschitz constant of final weights. The growth of these constants is bounded for kernels in $\mathcal{T}(\mathcal{G})$ by Lemma 11.7. For non-kernel nodes of $\mathcal{T}(\mathcal{G})$, we use the careful analysis of Lemma 10.10, detailed in Chapter 10.1.5.

The overall time complexity of this method is triply-exponential in the size of the input game and polynomial in $1/\varepsilon$.

[1] Indeed, inf and sup are 1-Lipschitz-continuous functions, and with a fixed play ρ, the mapping $\mathsf{wt}_\mathsf{t} \mapsto \mathsf{wt}_\Sigma(\rho) + \mathsf{wt}_\mathsf{t}(\mathsf{last}(\rho))$ is 1-Lipschitz-continuous.

11.4. Example of an execution of the approximation schema

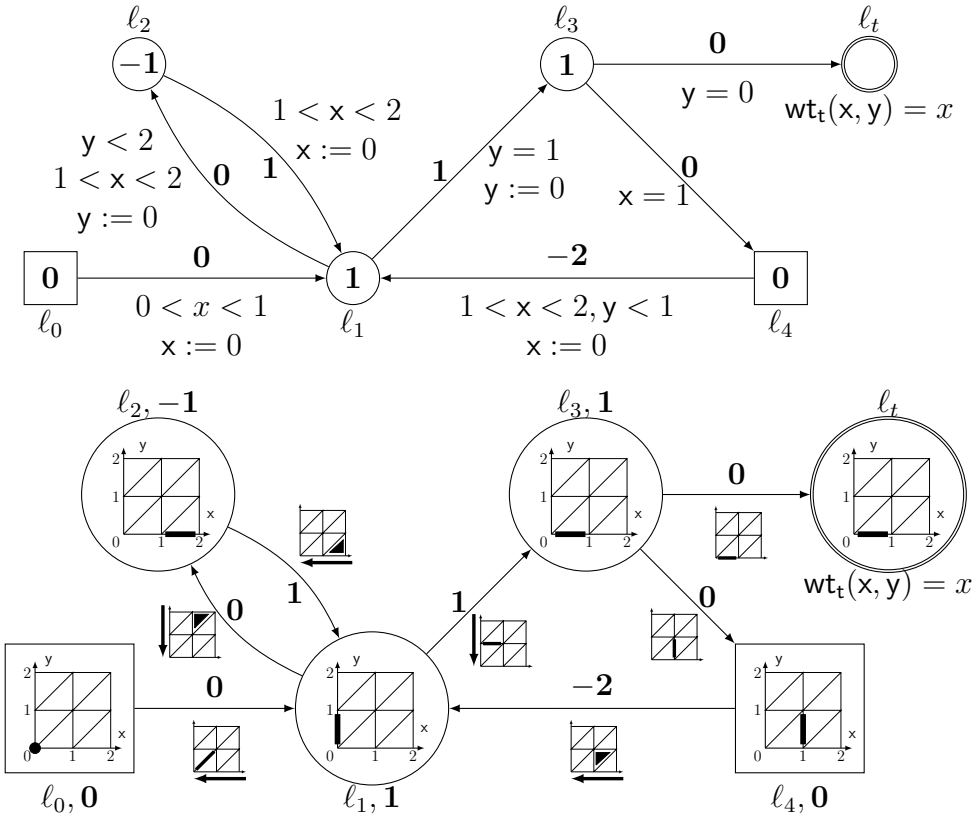


Figure 11.2.: A weighted timed game $\mathcal{G}$ with two clocks x and y, and the portion of its region game $\mathcal{R}(\mathcal{G})$ accessible from configuration $(\ell_0, (0,0))$. Locations of Min (resp. Max) are depicted as circles (resp. squares). The states of $\mathcal{R}(\mathcal{G})$ are labeled by their associated region, location and weight, and edges are labeled by a representation of their guards and resets. Since each location ℓ of $\mathcal{G}$ leads to a unique state (ℓ, r) of $\mathcal{R}(\mathcal{G})$, we will refer to states by their associated location label.

We are given the WTG $\mathcal{G}$ in Figure 11.2 and $\varepsilon \in \mathbb{Q}_{>0}$, and want to compute an ε-approximation of its value in location ℓ_0 for the valuation (x=0, y=0), denoted $\mathsf{Val}_{\mathcal{G}}(\ell_0, (0,0))$. In this example, we will use ε=15 because the computations would not be readable with a smaller precision. We also chose an example where $\mathcal{R}(\mathcal{G})$ is isomorphic to $\mathcal{G}$ for readability reasons. $\mathcal{R}(\mathcal{G})$ contains one SCC $\{\ell_1, \ell_2, \ell_3, \ell_4\}$, made of two simple cycles:

- $\mathsf{p}_1 = \ell_1 \to \ell_2 \to \ell_1$ is a positive cycle (all plays following p_1 have cumulated weight in the interval $(1, 3)$),

- and $\mathsf{p}_2 = \ell_1 \to \ell_3 \to \ell_4 \to \ell_1$ is a 0-cycle (all plays following p_2 have cumulated weight 0). This can be checked by Lemma 8.3.

Therefore, $\mathcal{R}(\mathcal{G})$ only contains non-negative SCCs and is almost-divergent. Since all states are in the attractor of Min towards L_{t}, all cycles are non-negative and the final weight function is bounded (on all reachable regions), there are no configurations in $\mathcal{R}(\mathcal{G})$ with value $+\infty$ or $-\infty$.

We let the kernel K be the sub-game of $\mathcal{R}(\mathcal{G})$ defined by p_2, and we construct a semi-unfolding $\mathcal{T}(\mathcal{G})$ of $\mathcal{R}(\mathcal{G})$ of equivalent value. Following Section 11.2.1 , we should unfold the game until every stopped branch contains a state seen at least $3|\mathcal{R}(\mathcal{G})|w_{\max} + 2w^{\mathsf{t}}_{\max} + 2 = 3 \times 3 \times 4 + 2 \times 1 + 2 = 40$ times. We will unfold with bound 4 instead of 40 for readability (it is enough on this example). Thus the infinite branch $(\ell_1\ell_2)^\omega$ is stopped when ℓ_1 is reached for the fourth time, as depicted in Figure 11.3.

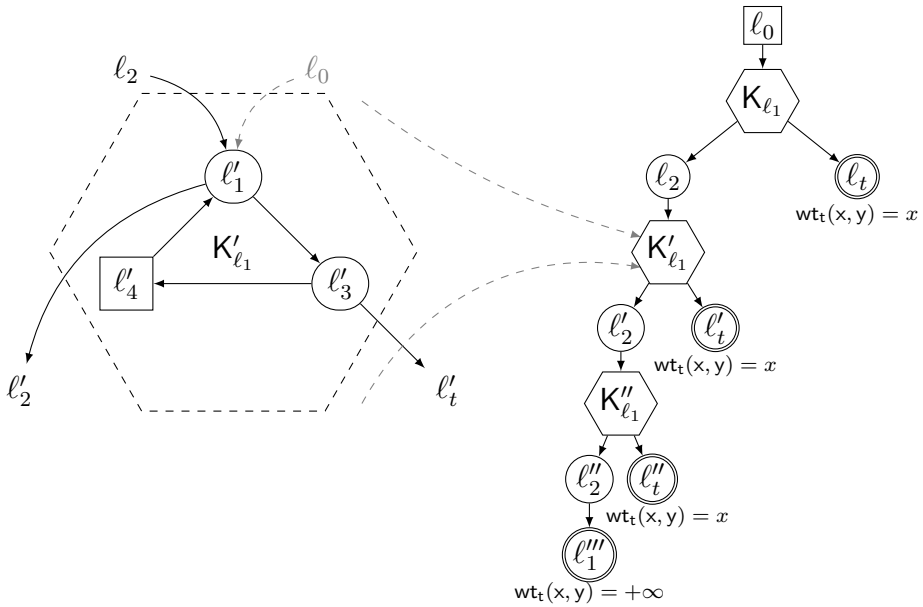


Figure 11.3.: The kernel K (with input state ℓ_1), and a semi-unfolding $\mathcal{T}(\mathcal{G})$ such that $\mathsf{Val}_{\mathcal{G}}(\ell_0, (0,0)) = \mathsf{Val}_{\mathcal{T}(\mathcal{G})}(\ell_0, (0,0))$. We denote by ℓ_i, ℓ'_i and ℓ''_i the locations in K, K' and K''.

Let us now compute an approximation of $\mathsf{Val}_{\mathcal{T}(\mathcal{G})}$. Let us first remove the states of value $+\infty$: ℓ'''_1 and ℓ''_2. Then, we start at the bottom and compute an $(\varepsilon/3)$-approximation of the value of ℓ''_1 in the game defined by K''_{ℓ_1} and its output edge to ℓ''_t. Following Section 11.3.1, we should use $N \geqslant 3(4+1)/\varepsilon$ and compute values in the $1/N$-corners game $\mathcal{C}_N(\mathsf{K}''_{\ell_1})$ in order to obtain an $(\varepsilon/3)$-approximation of the value function. For $\varepsilon = 15$ we will use $N = 1$ (in this case the computation happens to be exact and would also hold with a small ε). We construct this corner game, and obtain the finite (untimed) weighted game in Figure 11.4.

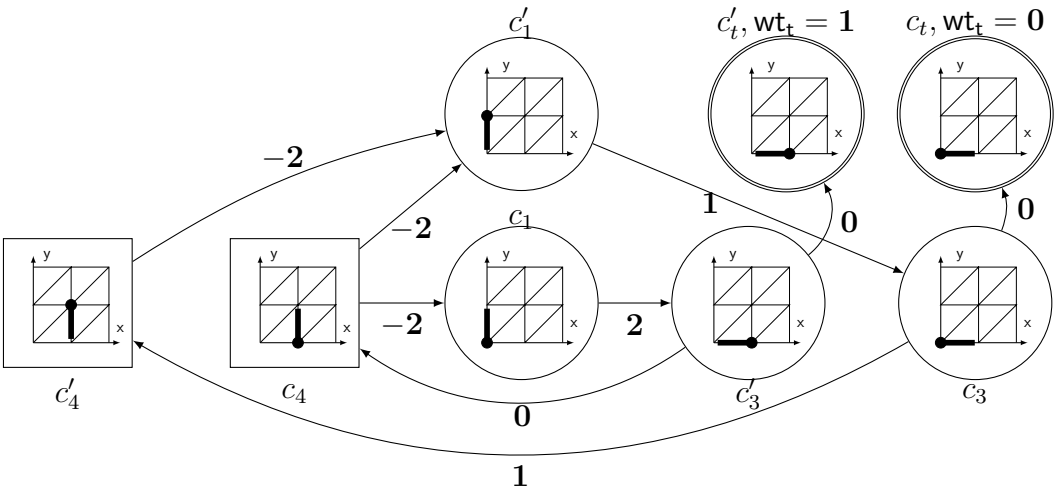


Figure 11.4.: The finite weighted game obtained from $\mathcal{C}_1(\mathsf{K}''_{\ell_1})$, where c_i and c'_i are the corners of ℓ''_i in $\mathcal{T}(\mathcal{G})$.

We can compute the values in this game to obtain $\mathsf{Val}(c'_1) = 1$ and $\mathsf{Val}(c_1) = 3$. We then define a value for every configuration in state ℓ''_1 by linear interpolation, obtaining:

$$(\mathsf{x}, \mathsf{y}) \mapsto 3 - 2\mathsf{y}\,.$$

This happens to be exactly $(\mathsf{x}, \mathsf{y}) \mapsto \mathsf{Val}_{\mathcal{T}(\mathcal{G})}(\ell''_1, (\mathsf{x}, \mathsf{y}))$ in this case, but would only be an $\varepsilon/3$-approximation of it in general. Now, we can compute an $\varepsilon/3$-approximation of $\mathsf{Val}_{\mathcal{T}(\mathcal{G})}(\ell'_2)$ with one step of value iteration, obtaining

$$(\mathsf{x}, \mathsf{y}) \mapsto \inf_{0<d<2-\mathsf{x}} (-1) \times d + 1 + 3 - 2(0 + d) = 3\mathsf{x} - 2\,.$$

The next step is computing an $\varepsilon/3$-approximation of the value of ℓ'_1 in the game defined by K'_{ℓ_1} and its output edges to ℓ'_t and ℓ'_2, of respective final weight functions $(\mathsf{x}, \mathsf{y}) \mapsto \mathsf{x}$ and $(\mathsf{x}, \mathsf{y}) \mapsto 3\mathsf{x} - 2$. This will give us a $2\varepsilon/3$-approximation of $\mathsf{Val}_{\mathcal{T}(\mathcal{G})}(\ell'_1)$.

Following Section 11.3.1 once again, we should use $N \geqslant 3(5+3)/\varepsilon$ and compute values in the $1/N$-corners game $\mathcal{C}_N(\mathsf{K}'_{\ell_1})$. For $\varepsilon = 15$ this gives $N = 2$ (which will once again keep the computation exact). We can construct a finite (untimed) weighted game as in Figure 11.4, and obtain a value for each 1/2-corner of state ℓ'_1:

- On the 1/2-region $(0 < \mathsf{y} < 1/2, \mathsf{x} = 0)$, corner $(0, 0)$ has value 2 and corner $(0, 1/2)$ has value 2.
- On the 1/2-region $(\mathsf{y} = 1/2, \mathsf{x} = 0)$, corner $(0, 1/2)$ has value 2.
- On the 1/2-region $(1/2 < \mathsf{y} < 1, \mathsf{x} = 0)$, corner $(0, 1/2)$ has value 2 and corner $(0, 1)$ has value 1.

From these results, we define a piecewise-linear function by interpolating the values of

corners on each 1/2-region, and obtain

$$(\mathsf{x},\mathsf{y}) \mapsto \begin{cases} 2 & \text{if } \mathsf{y} \leqslant 1/2 \\ 3-2\mathsf{y} & \text{otherwise} \end{cases}$$

as depicted in Figure 11.5.

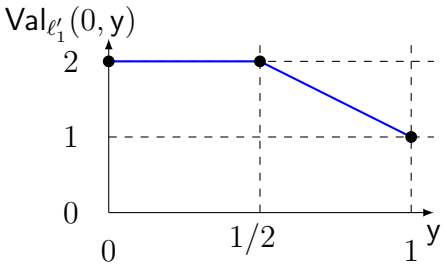


Figure 11.5.: The value function $(\mathsf{x},\mathsf{y}) \mapsto \mathsf{Val}_{\mathcal{T}(\mathcal{G})}(\ell_1', (\mathsf{x},\mathsf{y}))$, projected on $\mathsf{x} = 0$. Black dots represent the values obtained for 1/2-corners using the corner-point abstraction.

This gives us a $2\varepsilon/3$-approximation of $(\mathsf{x},\mathsf{y}) \mapsto \mathsf{Val}_{\mathcal{T}(\mathcal{G})}(\ell_1', (\mathsf{x},\mathsf{y}))$ (in fact exactly $\mathsf{Val}_{\mathcal{T}(\mathcal{G})}(\ell_1')$). Now, we can compute a $2\varepsilon/3$-approximation of $\mathsf{Val}_{\mathcal{T}(\mathcal{G})}(\ell_2)$ on region $(1 < \mathsf{x} < 2, \mathsf{y} = 0)$ with one step of value iteration, obtaining :

$$(\mathsf{x},\mathsf{y}) \to \inf_{0<d<2-\mathsf{x}} \begin{cases} 3-d & \text{if } d \leqslant 1/2 \\ 4-3d & \text{otherwise} \end{cases} = \begin{cases} 3\mathsf{x}-2 & \text{if } \mathsf{x} \leqslant 3/2 \\ \mathsf{x}+1 & \text{otherwise} \end{cases}$$

Then, we need to compute an $\varepsilon/3$-approximation of the value of ℓ_1 in the game defined by K_{ℓ_1} and its output edges to ℓ_t and ℓ_2, of respective final weight functions $(\mathsf{x},\mathsf{y}) \mapsto \mathsf{x}$ and $(\mathsf{x},\mathsf{y}) \mapsto 3\mathsf{x}-2$ if $\mathsf{x} \leqslant 3/2; \mathsf{x}+1$ otherwise. This will give us an ε-approximation of $\mathsf{Val}_{\mathcal{T}(\mathcal{G})}(\ell_1)$.

Following Section 11.3.1 one last time, we should use $N \geqslant 3(5+3)/\varepsilon$ and compute values in the $1/N$-corner game $\mathcal{C}_N(\mathsf{K}_{\ell_1})$. This time, let us use $N = 3$ to showcase an example where the computed value is not exact. We can construct a finite (untimed) weighted game as in Figure 11.4, and obtain a value for each 1/3-corner of state ℓ_1'. From these results, we define a piecewise-linear function by interpolation, as depicted in Figure 11.6.

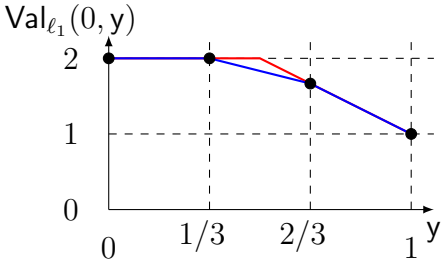


Figure 11.6.: The value function $(\mathsf{x},\mathsf{y}) \mapsto \mathsf{Val}_{\mathcal{T}(\mathcal{G})}(\ell_1,(\mathsf{x},\mathsf{y}))$, projected on $\mathsf{x} = 0$, is depicted in red. Black dots represent the values obtained for 1/3-corners using the corner-points abstraction, and the derived approximation of the value function is depicted in blue

Finally, from this ε-approximation of $\mathsf{Val}_{\mathcal{T}(\mathcal{G})}(\ell_1)$, we can compute an ε-approximation of $\mathsf{Val}_{\mathcal{T}(\mathcal{G})}(\ell_0)$ using one step of value iteration, and conclude. On our example this ensures

$$\mathsf{Val}_{\mathcal{T}(\mathcal{G})}(\ell_0,(0,0)) = \sup_{0<d<1} \mathsf{Val}_{\mathcal{T}(\mathcal{G})}(\ell_1,(0,d)) \in [2-\varepsilon, 2+\varepsilon]\,.$$

11.5. Symbolic approximation algorithm

The previous approximation result suffers from several drawbacks. It relies on the SCC decomposition of the region automaton. Each of these SCCs have to be analysed in a sequential way, and their analysis requires an a priori refinement of the granularity of regions. This approach is thus not easily amenable to implementation. We instead prove in this section that the symbolic approach based on the value iteration paradigm, *i.e.* the computation of iterates of the operator $\mathcal{F}$ recalled in page 126, is an approximation schema. This is stated in Theorem 9.3, for which we now sketch a proof in this section.

Notice that configurations with value $+\infty$ are stable through value iteration, and do not affect its other computations. Since Theorem 9.3 assumes the absence of configurations of value $-\infty$, we will therefore consider in the following that all configurations have finite value in $\mathcal{G}$.

Consider first a game $\mathcal{G}$ that is a kernel. By the results of Section 11.3.1, there exists an integer N such that solving the untimed weighted game $\Gamma_N(\mathcal{G})$ computes an $\varepsilon/2$-approximation of the value of $1/N$ corners. Using the results of [BGHM16] for untimed weighted games, we know that those values are obtained after a finite number of steps of (the untimed version of) the value iteration operator. More precisely, if one considers a number of iterations

$$P = |L||\mathsf{Reg}_N(\mathcal{X},M)|(|\mathcal{X}|+1)(2(|L||\mathsf{Reg}_N(\mathcal{X},M)|(|\mathcal{X}|+1)-1)w_{\max}+1)\,,$$

then $\mathsf{Val}^P_{\Gamma_N(\mathcal{G})}((\ell,r,\mathsf{v}),\mathsf{v}) = \mathsf{Val}_{\Gamma_N(\mathcal{G})}((\ell,r,\mathsf{v}),\mathsf{v})$. From this observation, we deduce the following property of P:

Lemma 11.8. *If $\mathcal{G}$ is a kernel with no configurations of infinite value, then $|\mathsf{Val}_{\mathcal{G}}(\ell,\nu) - \mathsf{Val}^P_{\mathcal{G}}(\ell,\nu)| \leqslant \varepsilon$ for all configurations (ℓ,ν) of $\mathcal{G}$.*

Proof. We already know that $\mathsf{Val}^P_{\Gamma_N(\mathcal{G})}((\ell,r,\mathsf{v}),\mathsf{v}) = \mathsf{Val}_{\Gamma_N(\mathcal{G})}((\ell,r,\mathsf{v}),\mathsf{v})$ for all configurations $((\ell,r,\mathsf{v}),\mathsf{v})$ of $\Gamma_N(\mathcal{G})$. Moreover, Section 11.3.1 ensures $|\mathsf{Val}_{\mathcal{G}}(\ell,\nu) - \mathsf{Val}_{\Gamma_N(\mathcal{G})}((\ell,r,\mathsf{v}),\mathsf{v})| \leqslant \varepsilon/2$ whenever ν is in the $1/N$-region r. Therefore, we only need to prove that $|\mathsf{Val}^P_{\mathcal{G}}(\ell,\nu) - \mathsf{Val}^P_{\Gamma_N(\mathcal{G})}((\ell,r,\mathsf{v}),\mathsf{v})| \leqslant \varepsilon/2$ to conclude. This is done as for Lemma 11.6, since Lemma 11.5 (that we need to prove Lemma 11.6) does not depend on the length of the plays ρ and ρ', and both runs reach the target state in the same step, *i.e.* both before or after the horizon of P steps. □

Once we know that value iteration converges on kernels, we can use the semi-unfolding of Section 11.2 to prove that it also converges on non-negative SCCs when all values are finite.

Lemma 11.9. *If $\mathcal{G}$ is a non-negative SCC with no configurations of infinite value, we can compute P_+ such that $|\mathsf{Val}_{\mathcal{G}}(\ell,\nu) - \mathsf{Val}^{P_+}_{\mathcal{G}}(\ell,\nu)| \leqslant \varepsilon$ for all configurations (ℓ,ν) of $\mathcal{G}$.*

Proof. Consider a non-negative SCC $\mathcal{G}$, a precision ε, and an initial configuration (ℓ_0,ν_0). Let $\mathcal{T}(\mathcal{G})$ be its finite semi-unfolding (obtained from the labelled tree T, as in Section 11.2.1), such that $\mathsf{Val}_{\mathcal{G}}(\ell_0,\nu_0) = \mathsf{Val}_{\mathcal{T}(\mathcal{G})}((\tilde{\ell}_0,r_0),\nu_0)$. Let α be the maximum number of kernels along a branch of T. Let P' be an integer such that for all kernels K in $\mathcal{T}(\mathcal{G})$, $|\mathsf{Val}_{\mathsf{K}}(\ell,\nu) - \mathsf{Val}^{P'}_{\mathsf{K}}(\ell,\nu)| \leqslant \varepsilon/\alpha$ for all configurations (ℓ,ν) of $\mathcal{G}$. We can find such a P' by using Lemma 11.8.

Create $\mathcal{T}'(\mathcal{G})$ from T by applying the method used to create $\mathcal{T}(\mathcal{G})$ but replace every kernel by its complete P'-unfolding instead. This implies that $\mathcal{T}'(\mathcal{G})$ is a tree, of bounded depth P (at most the depth of T times P'). Then

$$|\mathsf{Val}_{\mathcal{T}(\mathcal{G})}((\tilde{\ell}_0,r_0),\nu_0) - \mathsf{Val}_{\mathcal{T}'(\mathcal{G})}((\tilde{\ell}_0,r_0),\nu_0)| \leqslant \varepsilon\,.$$

This holds because the value function is 1-Lipschitz-continuous with regard to the final weight function, so imprecision builds up additively.

Consider now $\mathcal{T}''(\mathcal{G})$ the (complete) unfolding of $\mathcal{R}(\mathcal{G})$ with unfolding depth P, where kernels are also unfolded. By construction, $\mathsf{Val}_{\mathcal{T}''(\mathcal{G})}((\tilde{\ell}_0,r_0),\nu_0) = \mathsf{Val}^P_{\mathcal{T}''(\mathcal{G})}((\tilde{\ell}_0,r_0),\nu_0)$. Then, we can prove that $\mathsf{Val}^P_{\mathcal{T}''(\mathcal{G})}((\tilde{\ell}_0,r_0),\nu_0) = \mathsf{Val}^P_{\mathcal{G}}(\ell_0,\nu_0)$ (same strategies at bounded horizon P), which implies $\mathsf{Val}_{\mathcal{R}(\mathcal{G}))}((\ell_0,r_0),\nu_0) \leqslant \mathsf{Val}_{\mathcal{T}''(\mathcal{G})}((\tilde{\ell}_0,r_0),\nu_0)$ (monotonicity of Val^k). By another monotonicity argument (because $\mathcal{T}''$ contains $\mathcal{T}'$ as a prefix), we can also prove $\mathsf{Val}_{\mathcal{T}''(\mathcal{G})}((\tilde{\ell}_0,r_0),\nu_0) \leqslant \mathsf{Val}_{\mathcal{T}'(\mathcal{G})}((\tilde{\ell}_0,r_0),\nu_0)$.

Bringing everything together we obtain $|\mathsf{Val}^P_{\mathcal{G}}(\ell_0,\nu_0) - \mathsf{Val}_{\mathcal{G}}(\ell_0,\nu_0)| \leqslant \varepsilon$. □

The idea is to unfold every kernel of the semi-unfolding game $\mathcal{T}(\mathcal{G})$ according to its bound in Lemma 11.8. More precisely, let α be the maximum number of kernels along one of the branches of $\mathcal{T}(\mathcal{G})$. In a bottom-up fashion, we can find for each kernel K in $\mathcal{T}(\mathcal{G})$ a bound P_{K} such that, for all configurations (ℓ,ν), $|\mathsf{Val}_{\mathsf{K}}(\ell,\nu) - \mathsf{Val}^{P_{\mathsf{K}}}_{\mathsf{K}}(\ell,\nu)| \leqslant \varepsilon/\alpha$. We thus unfold K in $\mathcal{T}(\mathcal{G})$ with depth up to P_{K}. After each kernel has been replaced this

way, $\mathcal{T}(\mathcal{G})$ is no longer a semi-unfolding, it is instead a (complete) unfolding of $\mathcal{R}(\mathcal{G})$, of a certain bounded depth P_+. This new bound P_+ is bounded by the former depth of $\mathcal{T}(\mathcal{G})$ to which is added α times the biggest bound P_{K} we need for the kernels. Now, $\mathcal{T}(\mathcal{G})$ is a tree of depth P_+ whose value at its root is ε-close to the value of $\mathcal{G}$. Finally, the value computed by $\mathsf{Val}_{\mathcal{G}}^{P_+}$ is bounded between $\mathsf{Val}_{\mathcal{G}}$ and $\mathsf{Val}_{\mathcal{T}(\mathcal{G})}$, which allows us to conclude.

The bound P_{K} for a kernel K depends linearly in Λ, the Lipschitz constant of value functions on locations of $\mathcal{T}(\mathcal{G})$ reachable from K. Once K has been replaced by its unfolding of depth P_{K}, the Lipschitz constant of the value function at the root of $\mathcal{T}(\mathcal{G})$ are thus bounded exponentially in Λ. This means that we ensure a bound for P_+ that is at most polynomial in $1/\varepsilon$, and that is of the order of a tower of α exponentials.

Proving the same property on non-positive SCCs requires more work, because the semi-unfolding gives final weight $-\infty$ to stop leaves, which does not integrate well with value iteration (initialisation at $+\infty$ on non-target states). However, by unfolding those SCCs slightly more (at most $|\mathcal{R}(\mathcal{G})|$ more steps), we can obtain the desired property with a similar bound P_-.

Lemma 11.10. *If $\mathcal{G}$ is a non-positive SCC with no configurations of infinite value, we can compute P_- such that $|\mathsf{Val}_{\mathcal{G}}(\ell,\nu) - \mathsf{Val}_{\mathcal{G}}^{P_-}(\ell,\nu)| \leqslant \varepsilon$ for all configurations (ℓ,ν) of $\mathcal{G}$.*

Proof. Consider a non-positive SCC $\mathcal{G}$, a precision ε, and an initial configuration (ℓ_0,ν_0). Let $\mathcal{T}(\mathcal{G})$ be its finite semi-unfolding (obtained from the labelled tree T, as in Section 11.2.1), such that $\mathsf{Val}_{\mathcal{G}}(\ell_0,\nu_0) = \mathsf{Val}_{\mathcal{T}(\mathcal{G})}((\tilde{\ell}_0, r_0),\nu_0)$.

We now change T, by adding a subtree under each stopped leaf: the complete unfolding of $\mathcal{R}(\mathcal{G})$, starting from the stopped leaf, of depth $|\mathcal{R}(\mathcal{G})|$. Let us name T^+ this unfolding tree. We then construct $\mathcal{T}^+(\mathcal{G})$ as before, based on T^+. Since we are in a non-positive SCC, $\mathcal{T}^+(\mathcal{G})$ must have final weight $-\infty$ on its stopped leaves. It is easy to see that $\mathsf{Val}_{\mathcal{G}}(\ell_0,\nu_0) = \mathsf{Val}_{\mathcal{T}^+(\mathcal{G})}((\tilde{\ell}_0, r_0),\nu_0)$ still holds (the proof was based on branches being long enough, and we increased the lengths). We now perform a small but crucial change: the final weight of stopped leaves in $\mathcal{T}^+(\mathcal{G})$ is set to $+\infty$ instead of $-\infty$. Trivially $\mathsf{Val}_{\mathcal{T}(\mathcal{G})}((\tilde{\ell}_0, r_0),\nu_0) \leqslant \mathsf{Val}_{\mathcal{T}^+(\mathcal{G})}((\tilde{\ell}_0, r_0),\nu_0)$ (we increased the final weight function). Let us prove that

$$\mathsf{Val}_{\mathcal{T}^+(\mathcal{G})}((\tilde{\ell}_0, r_0),\nu_0) \leqslant \mathsf{Val}_{\mathcal{T}(\mathcal{G})}((\tilde{\ell}_0, r_0),\nu_0)\,.$$

For a fixed $\eta > 0$, consider σ_{Min} a η-optimal strategy for player Min in $\mathcal{T}(\mathcal{G})$. Let us define σ^+_{Min}, a strategy for Min in $\mathcal{T}^+(\mathcal{G})$, by making the same choice as σ_{Min} on the common prefix tree, and once a node that is a stopped leaf in $\mathcal{T}(\mathcal{G})$ is reached, we switch to a memoryless attractor strategy of Min towards target states. Consider any strategy σ^+_{Max} of Max in $\mathcal{T}^+(\mathcal{G})$, and let σ_{Max} be its projection in $\mathcal{T}(\mathcal{G})$. Let ρ^+ denote the (maximal) play

$$\mathsf{play}_{\mathcal{T}^+(\mathcal{G})}(((\ell_0, r_0),\nu_0), \sigma^+_{\mathsf{Min}}, \sigma^+_{\mathsf{Max}}))\,,$$

and ρ be $\mathsf{play}_{\mathcal{T}(\mathcal{G})}(((\ell_0, r_0),\nu_0), \sigma_{\mathsf{Min}}, \sigma_{\mathsf{Max}}))$. By construction, ρ^+ does not reach a stopped leaf in $\mathcal{T}^+(\mathcal{G})$. If the play ρ^+ stays in the common prefix tree of T and T^+, then $\rho = \rho^+$, and

$$\mathsf{wt}_{\mathcal{T}^+(\mathcal{G})}(\rho^+) \leqslant \mathsf{Val}_{\mathcal{T}(\mathcal{G})}((\tilde{\ell}_0, r_0),\nu_0) + \eta\,.$$

If it does not, then ρ^+ has a prefix that reaches a stopped leaf in $\mathcal{T}(\mathcal{G})$: this must be ρ. This implies (see Lemma 11.4) that

$$\mathsf{wt}_{\mathcal{T}^+(\mathcal{G})}(\rho^+) < -|\mathcal{R}(\mathcal{G})|w_{\max} - w^{\mathsf{t}}_{\max} \leqslant \mathsf{Val}_{\mathcal{T}(\mathcal{G})}((\tilde{\ell}_0, r_0), \nu_0)\,.$$

Since this holds for all $\eta > 0$, we proved $\mathsf{Val}_{\mathcal{T}^+(\mathcal{G})}((\tilde{\ell}_0, r_0), \nu_0) \leqslant \mathsf{Val}_{\mathcal{T}(\mathcal{G})}((\tilde{\ell}_0, r_0), \nu_0)$, which finally implies that the two values are equal.

Then, we can follow the proof of Lemma 11.9 (with T^+ and $\mathcal{T}^+(\mathcal{G})$) in order to conclude. □

Now, if we are given an almost-divergent game $\mathcal{G}$ and a precision ε, we can add the bounds for value iteration obtained from each SCC by Lemmas 11.9 and 11.10, and obtain a final bound P such that for all $k \geqslant P$, $\mathsf{Val}^k_{\mathcal{G}}$ is an ε-approximation of $\mathsf{Val}_{\mathcal{G}}$.

11.5.1. Discussion

Overall, this leads to an upper bound complexity that is polynomial in $1/\varepsilon$ and of the order of a tower of n exponentials, with n polynomial in the size of the input WTG. However, we argue that this symbolic procedure is more amenable to implementation than the previous approximation schema. First, it avoids the three already mentioned drawbacks (SCC decomposition, sequential analysis of the SCCs, and refinement of the granularity of regions) of the previous approximation schema. Then, it allows one to directly launch the value iteration algorithm on the game $\mathcal{G}$, and we can stop the computation whenever we are satisfied enough by the approximation computed: however, there are no guarantees whatsoever on the quality of the approximation before the number of steps P given above. Finally, this schema allows one to easily obtain an almost-optimal strategy with respect to the computed value.

If $\mathcal{G}$ is not guaranteed to be free of configurations of value $-\infty$, then we must first perform the SCC decomposition of $\mathcal{R}(\mathcal{G})$, and, as $\mathcal{G}$ is almost-divergent, identify and remove regions whose value is $-\infty$, by Lemma 11.1. Then, we can apply the value iteration algorithm.

As a final remark, notice that our correctness proof strongly relies on Section 11.3.1, and thus would not hold with the approximation schema of [BJM15] (which does not preserve the continuity on regions of the computed value functions, in turn needed to define final weights on $1/N$-corners).

Conclusion

In this thesis we presented new results for several controller synthesis problems on timed automata, that we now summarize, while mentionning possible future research directions and open problems.

Robustness

In Part II, we studied the qualitative robust controller synthesis problem, that asks if a Büchi objective can be guaranteed, in a timed automaton, against infinitesimal perturbations of delays. We developed a zone-based solution to this problem, that relies on computing and comparing the reachability relations of paths in the timed automaton. We complemented our theoretical results by a prototype tool and a case study, that illustrates the application of robust controller synthesis in small or moderate size problems. Our prototype relies on the standard DBM libraries, that we use with twice as many clocks to store the constraints of the normalised constraint graphs encoding reachability relations.

In order to scale to larger models an interesting future direction is to study extrapolation operators, and understanding whether results from reachability analysis on timed automata [BBFL03, BBLP04, HSW11] can be adapted to the robustness setting. This would also be interesting from a theoretical point of view, as it could remove the need for bounded clocks. This seems to be a challenging task, and we plan to start by considering the integration of those operators in the computation of reachability relations. This would be of independent interest, as an abstraction-compatible compositional aproach to reachability analysis of timed systems.

We also plan to explore other ways to improve performance: different strategies can be adopted for the nested forward analysis, switching between the two modes using heuristics, a parallel implementation, etc.

Moreover, we took interest in a more quantitative problem, that asks what is the greatest admissible perturbation for controller. The methods developed for the qualitative problem do not extend to this setting, as we heavily relied on reasoning on arbitrarily small perturbations *e.g.* for Lemma 5.2. However, if the qualitative problem is satisfied, our symbolic algorithm finds a lasso where controller wins the perturbation game for some maximal perturbation value $\delta > 0$. One could then use the techniques of Chapter 6 to determine the greatest perturbation against which this strategy for controller holds. Controller may be able to resist a greater maximal perturbation with another strategy, where the choice of what edge to follow depends on the perturbations. Indeed, by the results of [SBMR13] such strategies could be safely ignored for the qualitative

problem, but this is unlikely to hold in the quantitative setting. Computing the largest admissible perturbation on a general timed automaton is therefore another intriguing perspective. The complexity (and even decidability) of an exact computation of this greatest perturbation is open, and would have to be compared in practice with a binary search approach that tries successively fixed values for δ, as the fixed-perturbation problem can be solved by reduction to Büchi timed games [CHP11, LLTW14].

Weighted timed games

Our study of weighted timed games in Part III belongs to a series of works that explore the frontier of decidability. We introduced the first decidable class of weighted timed games with arbitrary weights and no restrictions on the number of clocks. We have given an approximation procedure for a larger class of weighted timed games, where the exact problem becomes undecidable. In addition, we proved the correctness of a symbolic approximation schema, that does not start by splitting exponentially every region, but only does so when necessary. We argue that this paves the way towards an implementation of value approximation for weighted timed games. Such tool would likely struggle with instances of moderate size, but could help with the design and testing of alternative approaches that trade theoretical guarantees with performance.

Another perspective is to extend this work to the concurrent setting, where both players play simultaneously and the shortest delay is selected. It should be noted that several known results on weighted timed games with non-negative weights [BCFL04, ABM04, BJM15] are stated in such a concurrent setting. We did not consider this setting in this work because concurrent WTGs are not determined, and several of our proofs rely on this property for symmetrical arguments (mainly to lift results of non-negative SCCs to non-positive ones).

A long-standing open problem is the approximation of weighted timed games, *i.e.* whether one can compute an arbitrarily close approximation of the value of a given game. We successfully solved this problem on the class of almost-divergent games, and introduced in Chapter 9 the slightly larger class of 0-isolated games. We do not have approximation results on this 0-isolated class, and as such it forms a natural intermediate step between the best known decidable class and the general case. The answer to this open question could also be negative, and there remain gaps in the complexity bounds obtained thus far for decidable classes. Therefore, pursuing better lower bounds could prove enlightening, and is another possible direction for future research.

Robust timed games

An interesting generalisation of our results on robustness it to consider timed games instead of timed automata. On top of perturbing delays picked by the controller, the environment could also choose some transitions to follow (for example in a turn-based fashion). The region-based approach of Chapter 4 can be extended to such a

setting [ORS14], so that solving the (qualitative) robust controller synthesis problem for a Büchi condition becomes EXPTIME. A purely symbolic method remains some distance away, as the matter would need to be settled without robustness as a first step (by adapting an on-the-fly algorithm for solving finite Büchi games to the timed setting).

Finally, one could consider bringing robustness issues to weighted timed games, by subjecting the controller to perturbations over delays. Under this semantics, the optimal reachability problem stays undecidable [BMS13], and related work considers very restricted classes when combining notions of optimality and robustness. For instance, the timed automata studied in [BBF+18] under optimality and robustness concerns are tree-shaped structures with one self-loop for each leaf.

Bibliography

[ABM04] Rajeev Alur, Mikhail Bernadsky, and P. Madhusudan. Optimal reachability for weighted timed games. In *Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP'04)*, volume 3142 of *Lecture Notes in Computer Science*, pages 122–133. Springer, 2004.

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[ALTP04] Rajeev Alur, Salvatore La Torre, and George J. Pappas. Optimal paths in weighted timed automata. *Theoretical Computer Science*, 318(3):297–322, 2004.

[AM99] Eugene Asarin and Oded Maler. As soon as possible: Time optimal control for timed automata. In *Hybrid Systems: Computation and Control*, volume 1569 of *Lecture Notes in Computer Science*, pages 19–30. Springer, 1999.

[BBBR07] Patricia Bouyer, Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin. On the optimal reachability problem of weighted timed automata. *Formal Methods in System Design*, 31(2):135–175, 2007.

[BBF+18] Giovanni Bacci, Patricia Bouyer, Uli Fahrenberg, Kim G. Larsen, Nicolas Markey, and Pierre-Alain Reynier. Optimal and robust controller synthesis using energy timed automata with uncertainty. In Bill W. Roscoe and Jan Peleska, editors, *Proceedings of the 22nd International Symposium on Formal Methods (FM'18)*, Lecture Notes in Computer Science, pages 203–221, Oxford, UK, July 2018. Springer. Best paper award.

[BBFL03] Gerd Behrmann, Patricia Bouyer, Emmanuel Fleury, and Kim G. Larsen. Static guard analysis in timed automata verification. In Hubert Garavel and John Hatcliff, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 254–270, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[BBL08] Patricia Bouyer, Ed Brinksma, and Kim G. Larsen. Optimal infinite scheduling for multi-priced timed automata. *Formal Methods in System Design*, 32(1):3–23, 2008.

[BBLP04] Gerd Behrmann, Patricia Bouyer, Kim G. Larsen, and Radek Pelánek. Lower and upper bounds in zone based abstractions of timed automata. In

Kurt Jensen and Andreas Podelski, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 312–326, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[BBM06] Patricia Bouyer, Thomas Brihaye, and Nicolas Markey. Improved undecidability results on weighted timed automata. *Information Processing Letters*, 98(5):188–194, 2006.

[BBR05] Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin. On optimal timed strategies. In *Proceedings of the Third international conference on Formal Modeling and Analysis of Timed Systems (FORMATS'05)*, volume 3829 of *LNCS*, pages 49–64. Springer, 2005.

[BCFL04] Patricia Bouyer, Franck Cassez, Emmanuel Fleury, and Kim G. Larsen. Optimal strategies in priced timed game automata. In *Proceedings of the 24th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'04)*, volume 3328 of *LNCS*, pages 148–160. Springer, 2004.

[BCR14] Romain Brenguier, Franck Cassez, and Jean-François Raskin. Energy and mean-payoff timed games. In *Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control (HSCC'14)*, pages 283–292. ACM, 2014.

[BDM+98] Marius Bozga, Conrado Daws, Oded Maler, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. Kronos: A model-checking tool for real-time systems. In Alan J. Hu and Moshe Y. Vardi, editors, *Computer Aided Verification (CAV 1998), Proceedings*, pages 546–550, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[BFH+01] Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim G. Larsen, Judi Romijn, and Frits W. Vaandrager. Minimum-cost reachability for priced timed automata. In *Proceedings of the 4th International Workshop on Hybrid Systems: Computation and Control (HSCC'01)*, volume 2034 of *Lecture Notes in Computer Science*, pages 147–161. Springer, 2001.

[BGH+15] Thomas Brihaye, Gilles Geeraerts, Axel Haddad, Engel Lefaucheux, and Benjamin Monmege. Simple priced timed games are not that simple. In *Proceedings of the 35th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'15)*, volume 45 of *LIPIcs*, pages 278–292. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2015.

[BGHM15] Thomas Brihaye, Gilles Geeraerts, Axel Haddad, and Benjamin Monmege. To reach or not to reach? Efficient algorithms for total-payoff games. In *Proceedings of the 26th International Conference on Concurrency Theory*

(CONCUR'15), volume 42 of *LIPIcs*, pages 297–310. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2015.

[BGHM16] Thomas Brihaye, Gilles Geeraerts, Axel Haddad, and Benjamin Monmege. Pseudopolynomial iterative algorithm to solve total-payoff games and min-cost reachability games. *Acta Informatica*, 2016.

[BGNK+14] Thomas Brihaye, Gilles Geeraerts, Shankara Narayanan Krishna, Lakshmi Manasa, Benjamin Monmege, and Ashutosh Trivedi. Adding negative prices to priced timed games. In *Proceedings of the 25th International Conference on Concurrency Theory (CONCUR'14)*, volume 8704, pages 560–575. Springer, 2014.

[BJM15] Patricia Bouyer, Samy Jaziri, and Nicolas Markey. On the value problem in weighted timed games. In *Proceedings of the 26th International Conference on Concurrency Theory (CONCUR'15)*, volume 42 of *Leibniz International Proceedings in Informatics*, pages 311–324. Leibniz-Zentrum für Informatik, 2015.

[BM83] Bernard Berthomieu and Miguel Menasche. An enumerative approach for analyzing time Petri nets. In R. E. A. Mason, editor, *Information Processing 83 – Proceedings of the 9th IFIP World Computer Congress (WCC'83)*, pages 41–46. North-Holland/IFIP, September 1983.

[BMR06] Patricia Bouyer, Nicolas Markey, and Pierre-Alain Reynier. Robust model-checking of linear-time properties in timed automata. In José R. Correa, Alejandro Hevia, and Marcos Kiwi, editors, *LATIN 2006: Theoretical Informatics*, pages 238–249, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[BMS13] Patricia Bouyer, Nicolas Markey, and Ocan Sankur. Robust weighted timed automata and games. In Víctor Braberman and Laurent Fribourg, editors, *Formal Modeling and Analysis of Timed Systems*, pages 31–46, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[Bou03] Patricia Bouyer. Untameable timed automata! In Helmut Alt and Michel Habib, editors, *STACS 2003*, pages 620–631, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[BT10] John S. Baras and George Theodorakopoulos. *Path Problems in Networks*. 2010.

[BY04] Johan Bengtsson and Wang Yi. *Timed Automata: Semantics, Algorithms and Tools*, volume 3098 of *Lecture Notes in Computer Science*, pages 87–124. Springer, 2004.

[CDF+05] Franck Cassez, Alexandre David, Emmanuel Fleury, Kim G. Larsen, and Didier Lime. Efficient on-the-fly algorithms for the analysis of timed games. In *CONCUR 2005*, volume 3653, pages 66–80. Springer-Verlag, 2005.

[CHP11] Krishnendu Chatterjee, Thomas A. Henzinger, and Vinayak S. Prabhu. Timed parity games: Complexity and robustness. *Logical Methods in Computer Science*, 7(4), 2011.

[CHR02] Franck Cassez, Thomas A. Henzinger, and Jean-François Raskin. A comparison of control problems for timed and hybrid systems. In Claire Tomlin and Mark R. Greenstreet, editors, *Proceedings of the 5th International Workshop on Hybrid Systems: Computation and Control (HSCC'02)*, volume 2289 of *Lecture Notes in Computer Science*, pages 134–148. Springer, 2002.

[CHVB18] Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors. *Handbook of Model Checking*. Springer, 2018.

[CLJ99] Hubert Comon-Lundh and Yan Jurski. Timed automata and the theory of real numbers. In *Proceedings of CONCUR'99*, volume 1664 of *Lecture Notes in Computer Science*, pages 242–257. Springer, 1999.

[Dil90] David L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Automatic Verification Methods for Finite State Systems (CAV 1989)*, volume 407 of *Lecture Notes in Computer Science*, pages 197–212. Springer, 1990.

[DPH07] Andrea D'Ariano, Marco Pranzo, and Ingo A. Hansen. Conflict resolution and train speed coordination for solving real-time timetable perturbations. *IEEE Transactions on Intelligent Transportation Systems*, 8(2):208–222, June 2007.

[DWDMR08] Martin De Wulf, Laurent Doyen, Nicolas Markey, and Jean-François Raskin. Robust safety of timed automata. *Formal Methods in System Design*, 33(1):45–84, 2008.

[GM08] Michel Gondran and Michel Minoux. Graphs, dioids and semirings. new models and algorithms. *Operations Research/ Computer Science Interfaces Series*, 41, 01 2008.

[HIJM13] Thomas Dueholm Hansen, Rasmus Ibsen-Jensen, and Peter Bro Miltersen. A faster algorithm for solving one-clock priced timed games. In *Proceedings of the 24th International Conference on Concurrency Theory (CONCUR'13)*, volume 8052 of *Lecture Notes in Computer Science*, pages 531–545. Springer, 2013.

[HKSW11] Frédéric Herbreteau, Dileep Kini, B. Srivathsan, and Igor Walukiewicz. Using non-convex approximations for efficient analysis of timed automata.

In Supratik Chakraborty and Amit Kumar, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011)*, volume 13 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 78–89, Dagstuhl, Germany, 2011. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[HOS16] Thomas A. Henzinger, Jan Otop, and Roopsha Samanta. Lipschitz robustness of timed i/o systems. In Barbara Jobstmann and K. Rustan M. Leino, editors, *Verification, Model Checking, and Abstract Interpretation*, pages 250–267, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[HPT19] Frédéric Herbreteau, Gerald Point, and Thanh-Tung Tran. Tchecker. http://www.labri.fr/perso/herbrete/tchecker/index.html, 2019.

[HS06] Thomas A. Henzinger and Joseph Sifakis. The embedded systems design challenge. In *FM 2006*, volume 4085 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2006.

[HS10] Frédéric Herbreteau and B. Srivathsan. Efficient on-the-fly emptiness check for timed Büchi automata. In *ATVA 2010*, volume 6252 of *Lecture Notes in Computer Science*, pages 218–232. Springer, 2010.

[HSTW16] Frédéric Herbreteau, B. Srivathsan, Thanh-Tung Tran, and Igor Walukiewicz. Why liveness for timed automata is hard, and what we can do about it. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016)*, volume 65 of *LIPIcs*, pages 48:1–48:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

[HSW10] Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. Efficient emptiness check for timed Büchi automata. In *Computer Aided Verification (CAV 2010), Proceedings*, volume 6174 of *Lecture Notes in Computer Science*, pages 148–161. Springer, 2010.

[HSW11] Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. Better abstractions for timed automata. *Proceedings - Symposium on Logic in Computer Science*, 10 2011.

[HSW12] Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. Efficient emptiness check for timed Büchi automata. *Formal Methods in System Design*, 40(2):122–146, 2012.

[Imm81] Neil Immerman. Number of quantifiers is better than number of tape cells. *Journal of Computer and System Sciences*, 22(3):384–406, 1981.

[Imm88] Neil Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17:935–938, 1988.

[JR11] Rémi Jaubert and Pierre-Alain Reynier. Quantitative robustness analysis of flat timed automata. In *Proceedings of the 14th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'11)*, volume 6604 of *Lecture Notes in Computer Science*, pages 229–244. Springer, 2011.

[JT07] Marcin Jurdziński and Ashutosh Trivedi. Reachability-time games on timed automata. In *Proceedings of the 34th International Colloquium on Automata, Languages and Programming (ICALP'07)*, volume 4596 of *LNCS*, pages 838–849. Springer, 2007.

[KBB+08] Leonid Khachiyan, Endre Boros, Konrad Borys, Khaled Elbassioni, Vladimir Gurvich, Gabor Rudolf, and Jihui Zhao. On short paths interdiction problems: Total and node-wise limited interdiction. *Theory of Computing Systems*, 43(2):204–233, 2008.

[Li09] Guangyuan Li. Checking timed Büchi automata emptiness using lu-abstractions. In *FORMATS 2009*, volume 5813 of *Lecture Notes in Computer Science*, pages 228–242. Springer, 2009.

[LLTW14] Kim G. Larsen, Axel Legay, Louis-Marie Traonouez, and Andrzej Wasowski. Robust synthesis for real-time systems. *Journal of Theoretical Computer Science (TCS)*, 515:96–122, 2014.

[LOD+13] Alfons Laarman, Mads Chr. Olesen, Andreas Engelbredt Dalsgaard, Kim Guldstrand Larsen, and Jaco van de Pol. Multi-core emptiness checking of timed Büchi automata using inclusion abstraction. In *Computer Aided Verification (CAV 2013), Proceedings*, volume 8044 of *Lecture Notes in Computer Science*, pages 968–983. Springer, 2013.

[LPY97] Kim G. Larsen, Paul Pettersson, and Wang Yi. Uppaal in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1):134–152, Dec 1997.

[Mat02] Jiri Matousek. *Lectures on Discrete Geometry*. Springer-Verlag, Berlin, Heidelberg, 2002.

[ORS14] Youssouf Oualhadj, Pierre-Alain Reynier, and Ocan Sankur. Probabilistic robust timed games. In Paolo Baldan and Daniele Gorla, editors, *CONCUR 2014 – Concurrency Theory*, pages 203–217, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[PGS17] Pavithra Prabhakar and Miriam García Soto. Formal synthesis of stabilizing controllers for switched systems. In *Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control (HSCC '17)*, pages 111–120, New York, NY, USA, 2017. ACM.

[PR89] Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. *Automata Languages and Programming*, 372:179–190, 01 1989.

[PS16] Pavithra Prabhakar and Miriam Garcia Soto. Counterexample guided abstraction refinement for stability analysis. In *Computer Aided Verification (CAV 2016), Proceedings, Part I*, pages 495–512, 2016.

[Pur00] Anuj Puri. Dynamical properties of timed automata. *Discrete Event Dynamic Systems*, 10(1-2):87–113, 2000.

[QSW17] Karin Quaas, Mahsa Shirmohammadi, and James Worrell. Revisiting reachability in timed automata. In *Proceedings of the 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2017)*. IEEE, 2017.

[Ros41] Barkley Rosser. Explicit bounds for some functions of prime numbers. *American Journal of Mathematics*, 63(1):211–232, 1941.

[RPV17] Nima Roohi, Pavithra Prabhakar, and Mahesh Viswanathan. Robust model checking of timed automata under clock drifts. In *Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control (HSCC'17)*, pages 153–162, New York, NY, USA, 2017. ACM.

[Rut11] Michał Rutkowski. Two-player reachability-price games on single-clock timed automata. In *Proceedings of the Ninth Workshop on Quantitative Aspects of Programming Languages (QAPL'11)*, volume 57 of *Electronic Proceedings in Theoretical Computer Science*, pages 31–46, 2011.

[Sav70] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.

[SBM11] Ocan Sankur, Patricia Bouyer, and Nicolas Markey. Shrinking Timed Automata. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011)*, volume 13 of *LIPIcs*, pages 90–102. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2011.

[SBMR13] Ocan Sankur, Patricia Bouyer, Nicolas Markey, and Pierre-Alain Reynier. Robust controller synthesis in timed automata. In *Proceedings of the 24th International Conference on Concurrency Theory (CONCUR'13)*, volume 8052 of *Lecture Notes in Computer Science*, pages 546–560. Springer, 2013.

[SHLG94] Viggo Stoltenberg-Hansen, Ingrid Lindström, and Edward R. Griffor. *Mathematical Theory of Domains*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1994.

[Sze88] Róbert Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26(3):279–284, 1988.

[TMM02] Salvatore La Torre, Supratik Mukhopadhyay, and Aniello Murano. Optimal-reachability and control for acyclic weighted timed automata. In *Proc. 2nd IFIP International Conference on Theoretical Computer Science (TCS'02*, pages 485–497. Kluwer, 2002.

[Tra16] Thanh-Tung Tran. *Verification of timed automata : reachability, liveness and modelling.* PhD thesis, University of Bordeaux, France, 2016.

[Tri09] Stavros Tripakis. Checking timed Büchi automata emptiness on simulation graphs. *ACM Trans. Comput. Log.*, 10(3):15:1–15:19, 2009.

[TYB05] Stavros Tripakis, Sergio Yovine, and Ahmed Bouajjani. Checking timed Büchi automata emptiness efficiently. *Formal Methods in System Design*, 26(3):267–292, 2005.

Synthèse symbolique de contrôleurs pour systèmes temporisés: robustesse et optimalité

Résumé : Le domaine de la synthèse réactive a pour objectif d'obtenir un système correct par construction à partir d'une spécification logique. Une approche classique consiste à se ramener à un jeu à somme nulle, où deux joueurs interagissent tour-à-tour dans un système de transitions, et à se demander si le joueur "contrôleur" peut garantir que son objectif sera rempli, et ce indépendamment des décisions du joueur "environnement". Nous étudions des spécifications temps-réel, modélisées par un automate temporisé équipé d'un objectif d'accessibilité ou de Büchi, et présentons des méthodes symboliques pour synthétiser des stratégies du contrôleur. Nos contributions concernent deux problématiques distinctes : on peut souhaiter que le contrôleur obtienne une stratégie robuste aux perturbations, ou bien le faire jouer de manière optimale dans un jeu pondéré.
Mots clés : Automates temporisés, Synthèse, Robustesse, Jeux pondéré

Symbolic controller synthesis for timed systems: robustness and optimality

Abstract: The field of reactive synthesis studies ways to obtain, starting from a specification, a system that is correct by construction. A classical approach models this setting as a zero-sum game played by two players on a transition system, and asks whether player controller can ensure an objective against any competing player environment. We focus on real-time specifications, modelled as timed automata with reachability or Büchi acceptance conditions, and present symbolic ways to synthesise strategies for the controller. We consider two problems, either restricting controller to robust strategies or aiming for optimal strategies in a weighted game setting.
Keywords: Timed automata, Synthesis, Robustness, Weighted games