Reachability in Timed Automata with Diagonal Constraints and Updates

by

Sayan Mukherjee

A thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science

to

Chennai Mathematical Institute

Submitted in November, 2021
Defended on March 30, 2022

Plot H1, SIPCOT IT Park, Siruseri,
Kelambakkam, Tamil Nadu 603103,
India

Supervisors

B. Srivathsan — Chennai Mathematical Institute, India
Paul Gastin — École Normale Supérieure Paris-Saclay, France

Doctoral Committee

Madhavan Mukund — Chennai Mathematical Institute, India
Patricia Bouyer-Decitre — École Normale Supérieure Paris-Saclay, France

Declaration

This thesis is a presentation of my original research work, carried out under the guidance of Prof. B Srivathsan at Chennai Mathematical Institute and Prof. Paul Gastin at École Normale Supérieure Paris-Saclay. This work has not formed the basis for the award of any degree, diploma, associateship, fellowship or other titles in Chennai Mathematical Institute or any other university or institution of higher education.

Sayan Mukherjee
March 30, 2022

In my capacity as the supervisor of the candidate's thesis, I certify that the above statements are true to the best of my knowledge.

B. Srivathsan
March 30, 2022

Paul Gastin
March 30, 2022

Acknowledgements

Although this thesis presents me as the sole author, I must mention everyone who made this possible – either by contributing to the content or by helping me be in a state where I could do whatever was required to produce this thesis, or both.

First and foremost, I would like to thank my supervisors – Srivathsan and Paul; without them I would not have been able to write this thesis. Having come from a non-computer science background, I required time to get accustomed to the subject first and then to research. I thank them for being patient and for always having time for me whenever I required it. I have thoroughly enjoyed our numerous discussions over these years. It is hard to summarize what all I have learnt from them, but, surely, whatever research acumen I have today is almost all because of them.

I would like to thank Paul for his encouragements and his wise advices – be it purely on the academic problem we were dealing with or in more general topics. These advices will be helpful no matter which career I end up having. I look forward to receiving more such advices in the years to come. I would also like to thank him for hosting our visits to Paris. The discussions we had during those visits paved the way for many results that constitute this thesis.

I have a lot to thank Srivathsan for. Firstly, for introducing me to the fields of Formal Verification and Timed Automata through his courses. Second and more importantly, for being a constant source of guidance for all these years. He always found the right words – strong enough to make me overcome my inertia; gentle enough to not knock me out – for me to overcome the obstacles that came along.

I would like to thank the reviewers for taking their time to read my thesis and give valuable feedback. I also thank Prof. Madhavan Mukund and Prof. Patricia Bouyer-Decitre for agreeing to be in my Doctoral Committee.

I am grateful to UMI ReLaX, Tata Consultancy Services - Innovation Labs (Pune, India) and Infosys Foundation (India) for their generous fundings that I have received during my PhD.

I would like to thank all the computer science faculties at CMI whose courses I have attended during my time in CMI. A major part of what I know about Theoretical Computer Science in general, is because of their teaching.

I could not have switched to Computer Science without the help of some of the teachers I had during my Master's at CMI. I would like to thank Sourish Das for being of immense help during those two years. His encouragement and guidance

were instrumental in me navigating my Master's and making my subsequent career choices. I thank Madhavan Mukund, firstly for his great courses that I have attended throughout my time at CMI, and secondly for the advices that he has gladly provided whenever I asked for it. Finally, I would like to thank Srivathsan for taking me as his student, had that not happened, most likely, I would not have done a PhD.

I thank the administrative staffs at CMI, especially, Rajeshwari, Ranjini and Sripathy – they made the administrative procedures extremely smooth and whenever I required anything they were always happy to help. I would also like to thank the canteen staff at CMI, in particular, Barun da. He has been here for my entire stay at CMI. I would fondly recall the great chats that we have had.

Coming to friends, I would first like to thank Govind for being a great friend and also for reading this thesis at various points and giving several suggestions. I cannot thank my flatmates Debodirna and Vishnu enough – it has been an absolute joy to have them as friends, as my flatmates and also for being a massive source of support during the initial months of lockdown. I would also like to thank the friends I made during my PhD at CMI, including Aashish, Anjali, Krishnendu, Malay, Pritthijit, Sayantani, Shanmugapriya and Sourav, among others. Having coffee together at CMI, going for food, going to watch movies and endless chats – their company made my stay at CMI a really enjoyable one.

Attending several conferences over the years have been a good experience because of the people I could attend these with – Adwitee, Keerthan, Sougata, Soumyajit and Sparsa. I would also like to thank Abhishek, Anirban and Ranadeep for helping me out during my visits to France.

I would like to thank the friends with whom I spent a major part of the initial couple of years of my PhD – Avinandan, Chayan, Rajat, Sayantan and several others. I will remember the great time that we spent together. Also the discussions I used to have with some of them about the courses we were attending together were of great help.

I thank the friends I was fortunate enough to make when I first came to CMI. Aritra, Arpan da, Bineet, Chiranjit da, Dwaipayan and Prabal – they helped me settle down in CMI and I will forever cherish the time we spent here.

I would also like to thank Neetal, Pranjal, Rajarshi, Ritam, Utsab and all other friends that I made during my time in CMI, for making my stay here memorable.

I would like to thank my friends from St. Xavier's College – Agnibesh, Arindam, Avishek, Debarpan, Nilasis and Prerona, for being the friends that they have been for all these years. I would also like to thank Arnab, for being a great friend and a great travel companion for the three years of BSc and also for popping up at Chennai – it has always been a pleasure to have him around.

Finally and most importantly, I would like to express my love and gratitude for my parents. No words can ever be enough to thank all the sacrifices that they have made over the years, to help me have the life that I have today. I would also like to thank my grandparents, especially my grandmother, and all my close relatives, for the unending affection that they have showered me with.

Abstract

Introduced by Alur and Dill, Timed Automata have been a popular tool for modelling real-time systems. Several syntactic extensions and restrictions of Timed Automata have been studied and various problems have been considered over these classes of automata. This thesis considers the classic reachability problem in Timed Automata, when the *guards* of the automata are allowed to contain diagonal constraints, that are expressions comparing the difference between two clocks with a non-negative integer, and also in the more general class called Updatable Timed Automata – introduced by Bouyer, Dufourd, Fleury and Petit – that allow transitions to set the values of clocks to arbitrary non-negative integers, or to the value of a clock, or even to the sum of the value of a clock and some integer. In contrast, Timed Automata only allow transitions to (re-)set values of clocks to 0. Diagonal constraints in Timed Automata provide exponential succinctness, whereas, updates increase the expressive power.

The aim of this thesis is to improve upon the existing algorithms for checking reachability in these two classes of automata. The restricted class of *diagonal-free* Timed Automata enjoys an efficient reachability algorithm, which is implemented in several tools, such as UPPAAL, KRONOS, and the more recent TChecker. This thesis aims to adapt this efficient algorithm to handle the larger classes as well.

In order to adapt the algorithm, two goals need to be achieved: (i) a suitable simulation relation needs to be constructed, and (ii) an algorithm needs to be devised for checking this relation between two zones.

This thesis discusses a new relation ($\sqsubseteq_G$), parameterized by a set G of atomic constraints over clocks. Not all choices for the parameter makes the relation a simulation relation. A fixpoint computation for constructing this parameter is proposed that ensures the relation becomes a simulation relation for Updatable Timed Automata and therefore for Timed Automata in particular, as well.

This thesis also describes an algorithm for checking $\sqsubseteq_G$ between two zones, when G is allowed to contain diagonals. In the diagonal-free case, this relation can be checked with the same complexity as that of the popular LU simulation relation. In the presence of diagonals, checking if this relation does not hold turns out to be NP-complete. A prototype implementation of this algorithm has been done inside TChecker. Some preliminary experiments performed using this prototype are reported in the thesis and these show gains over existing methods for handling diagonal constraints and updates.

List of publications

This thesis is based on the following publications.

- Paul Gastin, Sayan Mukherjee, and B. Srivathsan. Reachability in Timed Automata with Diagonal Constraints. In CONCUR 2018, volume 118 of LIPIcs, pages 28:1 – 28:17. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.
- Paul Gastin, Sayan Mukherjee, and B. Srivathsan. Fast Algorithms for Handling Diagonal Constraints in Timed Automata. In CAV 2019, volume 11561 of LNCS, pages 41 – 59, Springer.
- Paul Gastin, Sayan Mukherjee, and B. Srivathsan. Reachability for Updatable Timed Automata Made Faster and More Effective. In FSTTCS 2020, volume 182 of LIPIcs, pages 47:1 – 47:17. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

Contents

Chapter 1

Introduction

A system is considered to be *safe*, if it can *never exhibit* an *unexpected* behaviour. It is imperative to ensure safety of systems for which safety is *critical*, before these are deployed. One way of checking safety of a system, is enumerating all possible executions of the system and checking if any unexpected behaviour is possible. However, in general, real-world systems are large and these have too many (possibly infinitely many) possible executions. Therefore, enumerating all possible executions of a system becomes too hard – possibly, impossible.

Model Checking [CES86], proposed by Clarke et al, is an automated technique to address this problem. In this method, first a system is modelled as an automaton and the desired property (for example, safety) is specified using a formula in some suitable logic. Checking if the system is safe translates to checking reachability in the product of the automaton modelling the system and the automaton corresponding to the *negation* of the desired property. If reachable, the path to a target state provides a sequence of behaviours of the system leading to a bad state (that does not satisfy the desired property). Whereas, if unreachable, no execution of the system violates the desired property and hence the system is safe.

Alur and Dill proposed Timed Automata [AD94] – that has been used to model real-time systems. This model extends finite-state automata with *clocks*. Clocks are non-negative real valued variables, measuring time. All clocks of a timed automaton start with the value 0 and then elapse at the same rate. The transitions in Timed Automata consist of two components – *guard* and *reset*. A *guard* is either a constraint of the form $x \sim c$, comparing a clock x with a (non-negative) constant c, or a constraint of the form $x - y \sim c$ comparing the difference between two clocks with a (non-negative) constant or a conjunction of these. The automaton can take a transition once the current values of the clocks satisfy the guard of the transition. The automaton can also wait at a state. Once the automaton takes a transition, every clock belonging to the *reset* (specified as a set of clocks) of the transition are set back to 0 and start elapsing again.

Several properties of real-time systems can be verified when the system is mod-

elled using timed automata or using some extension of it. Various tools have been developed for this purpose, such as, UPPAAL [LPY97], KRONOS [Yov97], HyTech [HHW97], RED [Wan04], Romeo [LRST09], PAT [SLDP09], LTS-min [KLM+15], TiAMo [BCM16], THETA [THV+17], IMITATOR [And21] and TChecker [HP].

Several extensions and restrictions of Timed Automata have been studied over the years, for example, Parametric Timed Automata [AHV93], Hybrid Automata [HKPV98], Updatable Timed Automata [BDFP00a], Timed Automata with additive constraints [BD00], Weighted Timed Automata [ATP01, BFH+01], Probabilistic Timed Automata [KNSS00], Pushdown Timed Automata [BER94], Timed Pushdown Automata [AAS12], Event Clock Automata [AFH99].

Several problems have also been considered in Timed Automata, including, reachability (or language emptiness) [AD94], liveness [TYB05, HSTW16], robustness [BMS13], reachability relations [CJ99, FQSW20]. A list of several well-studied decision problems can be found in [AM04]. This thesis deals with the classic reachability problem in Timed Automata and in Updatable Timed Automata.

Apart from the technique that will be discussed throughout this thesis, various different techniques have been studied for the reachability problem in Timed Automata, such as the works in [BJLY98, NMA+02, GHSW19, RSM19].

When a real-time system is modelled as a timed automaton, the safety of the system translates to checking reachability in the timed automaton modelling it. The *reachability problem* asks: given a timed automaton $\mathcal{A}$ (modelling a given real-time system) and a state q of the automaton (representing bad behaviour), can the automaton, starting from an initial state, take a sequence of transitions and reach the state q? This problem was shown to be PSPACE-complete [AD94]. The tools, such as, UPPAAL [LPY97] and TChecker [HP], implement optimized algorithms for the reachability problem.

Diagonal constraints are expressions of the form $x-y \sim c$, where x, y are clocks, $\sim \in \{<, \leq, \geq, >\}$ and $c \in \mathbb{N}$. Diagonal constraints do not add any expressive power to Timed Automata [AD94], that is, every system that can be modelled using Timed Automata (containing diagonal constraints) can also be modelled using only *diagonal-free* Timed Automata. Given a timed automaton containing diagonal constraints, the diagonal constraints can be removed from the automaton keeping the *timed language* of the automaton unchanged. This implies, the reachability problem in the original automaton can be reduced to another reachability problem in the diagonal-free equivalent automaton.

Theorem 1.1 ([BDGP98])**.** *For every timed automaton $\mathcal{A}_d$ containing diagonal constraints, there exists a timed automaton $\mathcal{A}_{nd}$ that does not contain diagonal constraints and has the same timed language as $\mathcal{A}_d$.*

However, given a timed automaton with diagonal constraints, the systematic removal of diagonal constraints as presented by Bérard et al. in [BDGP98] results in an exponentially larger automaton without diagonal constraints. Moreover, it was shown by Bouyer and Chevalier in [BC05], that there exists a timed automaton with diagonal constraints $\mathcal{A}_d$, of size $\mathcal{O}(n^2.\log n)$, for which, every timed automaton

without diagonal constraints having the same timed language, is of size at least $\mathcal{O}(2^n)$. This exponential blowup is therefore, unavoidable in the worst case.

Theorem 1.2 ([BC05])**.** *Timed automata with diagonal constraints are exponentially more succinct than timed automata without diagonal constraints.*

Therefore, using diagonal constraints while modelling a system as a timed automaton, can yield exponentially smaller models.

Updates generalize resets. To recall, transitions in timed automata contain resets. These are sets of clocks associated with transitions. Once a transition is taken, every clock that belongs to its reset set is set back to 0. Instead of only resetting clocks to 0, updates allow clocks to be set to arbitrary (non-negative) integers, or even to a sum of the current value of a clock and an integer. Introduced by Bouyer et al. in [BDFP00a, BDFP00b, BDFP04], Updatable Timed Automata (UTA) extend the class of Timed Automata, by allowing updates in place of resets in its transitions. Checking reachability in UTA is undecidable in general. However, several decidable subclasses of UTA have also been studied [BDFP04].

Fersman et al. have studied a subclass of UTA in [FKPY07] – called Timed Automata with Bounded Subtraction – containing some of the updates as well as diagonal constraints. They used this class to model preemptive scheduling. This further highlights the importance of considering updates and diagonal constraints.

Using diagonal constraints and updates provide advantages while trying to model a real-time system – the former by yielding smaller models and the latter by providing more expressive power. However, these have not been used in practice as much as the restricted class of diagonal-free Timed Automata has been. This is perhaps because diagonal-free Timed Automata enjoy an efficient reachability algorithm. A brief overview of how reachability is checked in diagonal-free Timed Automata is described below. This will help in reading the rest of this chapter.

The main hurdle in checking reachability in Timed Automata is the fact that clocks range over (non-negative) *real* numbers – an uncountably infinite set. This makes the configuration space of Timed Automata also uncountably infinite. Therefore, enumerating the entire configuration space becomes impossible. In fact, the problem lies with the number of *valuations* (these are functions associating values to clocks). Working with individual valuations becomes impossible and therefore a need arises to work with collections of valuations instead. Two such collections exist – *regions* and *zones*. Since the number of regions is very large, region based algorithm fails to become useful and hence a zone based algorithm is instead used in practice. This algorithm, given a (diagonal-free) timed automaton, builds its *zone graph*. The nodes of this graph are pairs of the form (q, Z) where q is a state of the input automaton and Z is a zone. In order to build this graph, given a node (q, Z) and a transition (outgoing from q) of the input automaton, a mechanism exists for computing the successor of the node with respect to the transition. If this graph is built only by continuously computing successors of every node – this method may

not terminate. To ensure termination, two operations have been studied – *extrapolations* and *simulations* – to decide which nodes' successors need to be computed and which nodes' successors need not. An extrapolation operator, given a zone, enlarges the zone. Whereas, simulation based methods keep the zones as they are, but checks a relation between two zones and infers when a node of the zone graph needs to be *explored* further and when not. Several extrapolation operators and simulation relations exist for diagonal-free Timed Automata. More details about regions, zones, extrapolations and simulations will be discussed in **Chapter 2**.

The aim of this thesis is to improve upon the existing algorithms for checking reachability in these two classes of automata. The restricted class of *diagonal-free* Timed Automata enjoys an efficient zone based reachability algorithm, which is implemented in several tools, such as UPPAAL [LPY97], KRONOS [Yov97], and the more recent TChecker [HP]. This thesis tries to adapt this efficient algorithm to make it work in the presence of diagonal constraints and updates.

1.1 Handling diagonal constraints and updates: existing methods

This section describes the available algorithms and tool support for checking reachability in Timed Automata in the presence of diagonal constraints and updates.

In the presence of diagonals reachability is checked in one of the three methods described below.

Remove the diagonals. Since diagonal constraints can always be removed (Theorem 1.1), given a timed automaton with diagonal constraints, it is possible to compute an *equivalent* timed automaton without diagonal constraints and then use the reachability algorithm available for diagonal-free Timed Automata on that. However, removing all the diagonals using the procedure of [BDGP98] results in a systematic exponential blowup. Due to Theorem 1.2, for certain examples it indeed may not even be possible to remove all the diagonals without having the exponential blowup. Another option is to remove only *active* diagonals, in the spirit of [DY96, BBFL03]. However, due to Theorem 1.2 it will still not be possible to avoid the exponential blowup in the worst case. Therefore, it might be beneficial to devise an algorithm that can avoid the need to remove the diagonals from the input automaton.

To recall, *valuations* are maps associating (non-negative) real numbers to clocks. Given a constraint φ and a valuation v, the valuation is said to *satisfy* the constraint, if when the clocks in φ are replaced with their values under v, the constraint evaluates to `true` and the valuation is said to *not satisfy* the constraint otherwise.

Zone splitting. Extrapolation of zones in the presence of diagonal constraints can lead to unsound procedures [Bou03, Bou04]. It may so happen that for a zone Z and a diagonal φ, there are no valuations in Z that satisfy φ, however the extrapolated zone ($\mathsf{Extra}(Z)$) contains valuations satisfying φ. This may make a transition enabled from $\mathsf{Extra}(Z)$ which is not enabled from Z. In [BY03] Bengtsson and Yi proposed a method to avoid this problem. The idea behind this method is to ensure that no zones appearing in the reachability procedure, should contain two valuations such that one valuation satisfies a diagonal whereas the other does not satisfy the diagonal. In order to preserve this property, the algorithm *splits* every zone (appearing in a successor of a node in the zone graph) based on each of the diagonals present in the input automaton and then extrapolates each of them, using a modified extrapolation operator (which is mentioned as a *normalization* operator). This algorithm succeeds in avoiding the exponential blowup in the number of states, which is faced while removing the diagonals, however, this suffers from a *mandatory* exponential blowup in the number of nodes of the computed zone graph.

Refinement based reachability. This method was proposed by Bouyer et al. in [BLR05]. In this method, the diagonal-free reachability algorithm is first run on the automaton containing diagonal constraints. If one of the final states is found to be reachable, it is checked whether the path leading to the final state is a spurious one or not. If found to be spurious, then it is because of some diagonal that was ignored. A method was proposed in the paper, to find these diagonals along the path. An equivalent automaton is then computed by removing these particular diagonal constraints. This ensures the spurious path is no longer possible and then the algorithm for diagonal-free Timed Automata is rerun on this new automaton (containing at least one less diagonal). This process is repeated until either the diagonal-free algorithm returns that no final state is reachable, or, a path to some final state is found and it is proved to be not a spurious one.

For checking reachability in a timed automaton containing diagonal constraints, the first two methods each consists of a step with exponential (in the number of diagonal constraints present in the automaton) cost – first one in the number of states in the resulting diagonal-free automaton and the second one in the number of zones computed. This is a drawback for adopting either of these methods in practice. The third method tries to overcome this by using the idea that there are only a few diagonals that makes the diagonal-free algorithm produce a spurious trace. Therefore, the number of diagonals that need to be removed may turn out to be small, resulting in only a sub-exponential blowup in the state space. However, it is unclear if the number of such *relevant* diagonal constraints (that need to be removed) are few in general. It is therefore desirable, to have an algorithm that can check reachability without removing the diagonals first and also by possibly avoiding adding exponentially many nodes in place of a single node in the zone graph. This is the first goal that this thesis tries to achieve.

Goal 1. Devise an algorithm for checking reachability in the presence of diagonal constraints that avoids the exponential blowup in both the number of states and the number of zones computed.

Since the reachability algorithm used in the state-of-the-art tools perform well for diagonal-free Timed Automata the aim is to make this algorithm suitable in the presence of diagonals as well. In order to achieve this aim, a suitable extrapolation operator or a simulation relation needs to be defined. Since Bouyer proved in [Bou03] that no extrapolation operator will be correct when diagonal constraints are present, the aim therefore is to design a simulation relation instead, that is correct for checking reachability in the presence of diagonal constraints. Once such a simulation relation is defined, in order to plug it into the reachability algorithm framework, it is also necessary to devise an algorithm for checking this simulation between two zones. This then will result in a reachability algorithm, as long as the simulation relation is finite. Therefore, Goal 1 reduces to two sub-goals.

Goal 1.1. Find a simulation relation $\sqsubseteq$ that is "correct" for checking reachability in the presence of diagonal constraints.

Goal 1.2. Devise an algorithm for checking $Z \sqsubseteq Z'$ for two zones Z, Z'.

Tool support for diagonals. UPPAAL can check reachability in the presence of diagonal constraints. The zone splitting algorithm described in [BY03] is used to handle diagonals in UPPAAL. The refinement based algorithm has also been implemented in UPPAAL by Reynier [Rey07]. TChecker cannot check reachability in the presence of diagonal constraints.

Computing local parameters. Defining simulation relations in terms of *local* parameters as proposed by Behrmann et al. in [BBFL03] is an important optimization that speeds up the zone enumeration. This optimization has been studied for diagonal-free Timed Automata, and for a restricted set of updates. It should however be noted that the local parameter computation was studied in the context of region-based abstractions. Later, the more efficient LU abstractions were proposed [BBLP06]. The extension of the local parameter computation in the context of LU abstractions has not been studied in the presence of diagonal constraints or updates. Current tools implement the combination of LU abstractions and local parameters only for diagonal-free Timed Automata, along with a restricted set of updates. While defining a simulation relation, the aim in thesis will be – (i) using local parameters and (ii) devising a static analysis for computing the parameter.

Checking reachability in the presence of updates is undecidable in general. Several syntactic subclasses of Updatable Timed Automata have been studied for which reachability has been shown to be decidable. Moreover, the complexity of the reachability problem in each such class has also been settled [BDFP00a, BDFP04]. Two kinds of algorithms exist for checking reachability in UTA.

Using regions. The decidability of reachability in the subclasses of UTA described in [BDFP04] are shown using regions. Therefore, these regions can be used to devise an algorithm for checking reachability. However, because regions are too many, this method is not practical.

Using zones. Bouyer gave a zone based algorithm in [Bou04] that uses a Closure operator – defined based on regions. Instead of adding a zone Z as it is, in the zone graph, the algorithm adds $\mathsf{Closure}(Z)$. This algorithm was shown to be correct for checking reachability for each of the decidable subclasses listed down in [BDFP04, Bou04]. However, the problem with this approach is, $\mathsf{Closure}(Z)$ need not be convex. Herbreteau et al. gave an algorithm in [HSW16] for checking $Z_1 \subseteq \mathsf{Closure}(Z_2)$, given two zones Z_1, Z_2. This removes the necessity of storing $\mathsf{Closure}(Z)$ and therefore avoiding the problem with non-convexity. Hence, in principle, this method can be implemented. However, current Timed Automata tools use *abstractions* based on *lower-upper bounds*, which are significantly more efficient than region based abstractions. No such abstractions are known in the presence of updates, with or without diagonal constraints. This sets up the second goal of the thesis.

Goal 2. Devise an efficient zone based reachability algorithm for Updatable Timed Automata.

Similar to the case with diagonals, the aim is to adapt the efficient zone based algorithm available for diagonal-free Timed Automata to the case with updates. Again, the main hurdle towards achieving this aim is to be able to define a simulation relation and an efficient algorithm for checking this simulation between two zones. These constitute the other two sub-goals that this thesis tries to achieve.

Goal 2.1. Define a simulation relation for Updatable Timed Automata.

Goal 2.2. Devise an algorithm for checking the simulation relation between two zones.

Tool support for updates. Both the tools UPPAAL and TChecker allow inputs to contain updates of the form $x := c$, where $c \in \mathbb{N}$ and $x := y$, where y is a clock. TChecker also allows update of the form $x := y + d$, where y is a clock and $d \in \mathbb{N}$. However, neither of these tools allow updates of the form $x := y - d$, where y is a clock and $d \in \mathbb{N}$. Updates of the form $x := x - d$ are useful for modelling preemptive scheduling, a central benchmark for the work to be presented in this thesis.

The next section provides an outline of the contributions of the thesis, along with the organization of the chapters.

1.2 Contributions of this thesis

This thesis proposes a relation $\sqsubseteq_G$, parameterized by a set of constraints (G) of the form $x \sim c$ and $x - y \sim c$, where x, y are clocks and c is an integer. This relation is first defined between two valuations and subsequently lifted to a relation between two zones (these are collections of valuations). The aim is to make $\sqsubseteq_G$ a simulation relation. Not all choices of G make $\sqsubseteq_G$ a simulation relation. A construction of this parameter is proposed that ensures the relation becomes a simulation relation for Updatable Timed Automata and therefore for Timed Automata in particular, as well – achieving Goal 2.1, the first of the two goals of this thesis.

Chapter 3 defines the relation $\sqsubseteq_G$ – initially between two valuations and subsequently between two zones. This chapter first shows that, when the set G does not contain diagonal constraints, $\sqsubseteq_G$ relates more valuations than the LU simulation relation defined by Behrmann et al. in [BBLP06] for diagonal-free Timed Automata. This further implies that $\sqsubseteq_G$ can potentially relate more zones than LU.

Now, to ensure $\sqsubseteq_G$ becomes a simulation relation for Updatable Timed Automata (and therefore for Timed Automata containing diagonal constraints as well) the parameter needs to be computed carefully. In the spirit of state specific bounds functions defined by Behrmann et al. in [BBFL03], instead of using a single set G, a family of sets $\{G(q) \mid q$ is a state of the updatable timed automaton$\}$ is used to define the relation. This chapter proposes a construction of these sets using a fixpoint computation. This computation is not guaranteed to terminate for every updatable timed automaton – Chapter 4 describes why this is expected and Chapter 5 shows why this is not a drawback. The parameter construction presented in this chapter is part of the publications [GMS19, GMS20]. Given an updatable timed automaton, a procedure for checking if this fixpoint computation terminates for that automaton is also provided in this chapter. It turns out, checking termination of this computation is PTIME if the constants in the automaton are encoded in unary, whereas, it is PSPACE-complete if the constants are encoded in binary. This termination checking procedure is described in the publication [GMS20].

Constructing the simulation relation achieves the first goal of the thesis, however, another crucial goal (Goal 2.2) remains. In order to use this simulation relation in the reachability algorithm, an algorithm needs to be devised that can check this relation between two zones.

Chapter 4 starts with proving that whenever the fixpoint computation described in Chapter 3 terminates, the resulting simulation relation is *finite*. A simulation relation being finite means: in every infinite sequence of zones, there exist two zones where one is simulated by the other. This implies, the reachability algorithm is guaranteed to terminate when it uses a *finite* simulation relation. Therefore, for every automaton, for which the fixpoint computation terminates, the resulting simulation relation $\sqsubseteq_G$ can be used to decide reachability. Since reachability is undecidable in general for Updatable Timed Automata, it is expected that the fixpoint computation does not terminate for all updatable timed automata.

This chapter then proceeds towards devising an algorithm for checking the relation $\sqsubseteq_G$ between two zones, for every finite set of atomic constraints G, and therefore, in particular, for the G computed using the fixpoint computation of Chapter 3.

First, an algorithm is provided for the case when G does not contain diagonal constraints – these sets occur when considering diagonal-free Timed Automata. It turns out, in this restricted case, this relation can be checked in quadratic time – this matches with the complexity of checking the highly efficient LU-simulation relation, as proved by Herbreteau et al. in [HSW16].

This chapter builds on this algorithm to devise a recursive algorithm for checking $\sqsubseteq_G$ between two zones, when G is also allowed to contain diagonal constraints. This algorithm (part of the publication [GMS19]) calls the algorithm developed for

the diagonal-free case (in this place, LU can also be used instead), at most exponentially (in the number of diagonal constraints present in G) many times. Some heuristics have been discussed that can help decide the relation with fewer diagonal-free checks. However, it is proved that checking the (non-)simulation between two zones is, in fact, NP-complete. This NP-hardness proof is adapted from a similar proof presented in the publication [GMS18].

Chapter 5 proves that the fixpoint computation (for constructing the parameter of the simulation relation $\sqsubseteq_G$) described in Chapter 3 terminates for every decidable subclass of Updatable Timed Automata discussed in [BDFP04]. This fixpoint computation is also proved to terminate for the subclass of Updatable Timed Automata – called, Timed Automata with Bounded Subtraction – which was used by Fersman et al. in [FKPY07] to model preemptive scheduling. Since whenever the fixpoint computation terminates, the reachability algorithm with this simulation relation $\sqsubseteq_G$ also terminates (proved in Chapter 4), for all these classes of automata, reachability can be checked with $\sqsubseteq_G$.

A final contribution of this thesis is a prototype implementation of $\sqsubseteq_G$ and the algorithm for checking this relation between two zones in the tool TChecker [HP].

Chapter 6 gives a brief overview of the existing structure of TChecker. It further provides a description about the implementations of the simulation relation defined in Chapter 3 and the algorithm devised in Chapter 4. This involved defining a new data structure to represent the parameter of the simulation relation, implementing the fixpoint computation and the algorithm for checking the relation between two zones. The chapter reports some preliminary experiments that have been performed using this implementation. These show improvements over the existing algorithms for handling diagonal constraints and updates.

Chapter 7 concludes the thesis by summarizing the results obtained and a few possible research directions to be explored in future.

Chapter 2

Preliminaries

This thesis considers the reachability problem in Timed Automata, described by Alur and Dill in [AD90, AD94] and also in the extended class called Updatable Timed Automata, introduced by Bouyer et al. in [BDFP00a, BDFP00b, BDFP04]. This chapter first recalls the syntax and semantics of these two models and then discusses the algorithm to check reachability in Timed Automata *without* diagonal constraints, used by the tools like UPPAAL [LPY97] and TChecker [HP].

Throughout this thesis, $\mathbb{N}$ will denote the set of all natural numbers (including 0), $\mathbb{Z}$ the set of all integers and $\mathbb{R}$ the set of all real numbers. Sometimes, subscripts of the form $\bowtie c$ where $\bowtie \in \{<, \leq, \geq, >\}$ and $c \in \mathbb{N}$ will be used to denote subsets of these three sets where every element belonging to the subset satisfies the bound $\bowtie c$. For example, $\mathbb{R}_{\geq 0}$ will denote the set of all real numbers that are non-negative.

2.1 Timed Automata

Timed Automata extend the model of finite-state automata with *clocks*. A **clock** is a non-negative real valued variable, used to measure time quantitatively. Clocks will be denoted using lowercase letters like $x, y, \ldots$ (possibly with subscripts). Uppercase X will generally be used to denote a set of clocks.

Atomic Constraints are of two types – non-diagonal constraints and diagonal constraints. A *non-diagonal constraint* is an expression bounding a single clock: it is of the form $x \triangleleft c$ or $c \triangleleft x$, where x is a clock, $\triangleleft \in \{<, \leq\}$ and $c \in \mathbb{N}$. That is, $x \triangleleft c$ is either the expression $x < c$ or $x \leq c$.

A *diagonal constraint* bounds the difference of two clocks: it is an expression of the form $x - y \triangleleft c$ or $c \triangleleft x - y$, where x, y both are clocks, $\triangleleft \in \{<, \leq\}$ and $c \in \mathbb{N}$.

Two other kinds of *trivial* atomic constraints are also considered: $\top$ denotes the expression true and $\bot$ denotes the expression false.

Constraints. A *constraint* is either an atomic constraint or a conjunction of atomic constraints. More formally, a *constraint* is an expression that gets generated by the following grammar:

$$\varphi ::= \top \mid \bot \mid x \lhd c \mid c \lhd x \mid x - y \lhd c \mid c \lhd x - y \mid \varphi \wedge \varphi \tag{2.1}$$

where x, y are clocks, $\lhd \in \{<, \leq\}$ and $c \in \mathbb{N}$. Note that, only *non-negative* integers appear in the constraints. This convention, however, does not result in any loss of generality, since every constraint with a negative integer can be rewritten as an *equivalent* constraint with a positive integer.

Given a set of clocks X, $\Phi(X)$ denotes the set of all constraints that can be generated from the grammar in 2.1 using the clocks present in X.

Timed Automata contain finitely many states and transitions between them. The transitions of timed automata are made up of two components:

guard: a *guard* is a constraint belonging to $\Phi(X)$. These are used to control the movement of the automaton,

reset: every transition of a timed automaton can *reset* the values of some of the clocks back to 0. A *reset* is represented using a set of clocks, once a transition is taken the value of every clock belonging to its reset set is made 0.

Remark 1. The original Timed Automata model described by Alur and Dill in [AD94] considered labelled transitions. These labels are not relevant for the problem to be considered in this thesis. Therefore, to keep the syntax simple, labels on transitions are ignored. In practice, these labels are called *events* (or, *actions*). A transition is *enabled* only when the corresponding event happens. When considering a *network of timed automata*, events are used to synchronize transitions of different component automata of the network.

Definition 2.1 (Timed Automata [AD94])**.** *A timed automaton $\mathcal{A}$ is defined by a tuple (Q, X, q_0, T, F), where Q is a finite set of states, X is a finite set of clocks, q_0 is the starting state, $T \subseteq Q \times \Phi(X) \times 2^X \times Q$ is a finite set of transitions and $F \subseteq Q$ is the set of all accepting states. Any transition $t \in T$ is of the form (q, g, R, q'), where q, q' are called the source and the target state, respectively, $g \in \Phi(X)$ is the guard of the transition and $R \subseteq X$ is the reset set.*

A *diagonal-free timed automaton* is a timed automaton where no diagonal constraint appears in a guard, that is, every guard of the automaton is either a non-diagonal constraint or a conjunction of non-diagonal constraints.

All clocks of a timed automaton start at the same time with the value 0, and then elapse at the same rate.

Example 2.1.1. Figure 2.1 depicts a timed automaton $\mathcal{A}_1 = (Q, X, q_0, T, F)$, with $Q = \{q_0, q_1\}$ being the set of all states, the set of all clocks $X = \{x, y\}$, two transitions $t_1 = (q_0, x = 1, \{x\}, q_0)$ and $t_2 = (q_0, y - x = 10, \{\}, q_1)$ and $F = \{q_1\}$. [1]

[1] The equality constraints used in the guards are shorthands for conjunctions of two relevant atomic constraints. For example, $x = 1$ is used to write the constraint $x \leq 1 \wedge 1 \leq x$.

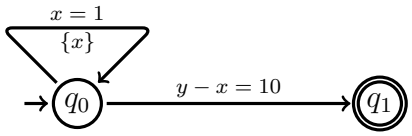


Figure 2.1: Example timed automaton $\mathcal{A}_1$

Valuations are maps assigning non-negative real values to clocks. More precisely, a *valuation* is a function $v\colon X \to \mathbb{R}_{\geq 0}$. Given a clock x and a valuation v, the notation $v(x)$ will be used to denote the value of the clock x under the valuation v. Given a set of clocks X, $\mathbb{R}^X_{\geq 0}$ is the set of all valuations over X.

Given a valuation $v \in \mathbb{R}^X_{\geq 0}$, a constant $\delta \in \mathbb{R}_{\geq 0}$ (called the *delay*) and a set of clocks R (*reset*), two valuations (to be frequently used in this thesis) $v + \delta$ and $[R](v)$ are defined in the following way:

- for all $x \in X$, $(v + \delta)(x) = v(x) + \delta$
- for all $x \in X$, $[R](v)(x) = \begin{cases} 0 & \text{if } x \in R \\ v(x) & \text{otherwise} \end{cases}$

Valuation satisfying a constraint. Given a valuation v and a constraint φ, the boolean expression $v(\varphi)$ is obtained by replacing the clocks present in φ with their values under the valuation v. That is, $v(\varphi)$ is defined as follows:

$$v(\varphi) = \begin{cases} v(x) \lhd c & \text{if } \varphi = x \lhd c \\ c \lhd v(x) & \text{if } \varphi = c \lhd x \\ v(x) - v(y) \lhd c & \text{if } \varphi = x - y \lhd c \\ c \lhd v(x) - v(y) & \text{if } \varphi = c \lhd x - y \\ v(\varphi_1) \wedge v(\varphi_2) & \text{if } \varphi = \varphi_1 \wedge \varphi_2 \end{cases} \tag{2.2}$$

A valuation v is said to satisfy a constraint φ, written as $v \models \varphi$, if the expression $v(\varphi)$ evaluates to true. Given a constraint φ, the notation $[\![\varphi]\!]$ will denote the set of all valuations that satisfy φ. Every valuation satisfies the atomic constraint $\top$ and no valuation satisfies $\bot$, that is, $[\![\top]\!] = \mathbb{R}^X_{\geq 0}$ and $[\![\bot]\!] = \emptyset$.

Configurations. A *configuration* of a timed automaton $\mathcal{A} = (Q, X, q_0, T, F)$ is a pair (q, v) where $q \in Q$ is a state of $\mathcal{A}$ and $v \in \mathbb{R}^X_{\geq 0}$ is a valuation. The configuration $(q_0, \mathbf{0})$ is called the *initial configuration* of $\mathcal{A}$, where $\mathbf{0}$ denotes the valuation mapping every clock (in X) to 0, that is, $\mathbf{0}(x) = 0$ for all $x \in X$.

A timed automaton can take a transition from a configuration (q, v) if the valuation v satisfies the guard of the transition. In such a case, the transition is said to be enabled from (q, v). For example, let v be a valuation with $v(x) = 2$ and consider a transition $t = (q, g, R, q')$ where $g = x \leq 2$, then t is enabled from the configuration (q, v). However, if the transition $t' = (q, g', R', q'')$ is considered with the guard $g' = x > 2$, then t' is not enabled from (q, v).

Definition 2.2 (Semantics of a timed automaton)**.** *The* semantics of a timed automaton $\mathcal{A} = (Q, X, q_0, T, F)$ *is specified using a transition system* $\mathcal{S}_\mathcal{A} = (\mathcal{S}, E)$ *where* $\mathcal{S}$ *is the set of all configurations of* $\mathcal{A}$*. The transitions in* E *are of two types:*

delay: *for every configuration* (q, v) *of* $\mathcal{A}$ *and every constant* $\delta \in \mathbb{R}_{\geq 0}$*, there exists a* delay *transition* $(q, v) \to^{\delta} (q, v + \delta)$ *in* E*,*

action: *for every transition* $t = (q, g, R, q') \in T$ *and every valuation* v *such that* $v \models g$*, there is an* action *transition* $(q, v) \to^{t} (q', v')$*, where* $v' = [R](v)$*.*

Remark 2. Due to the fact that the clocks range over $\mathbb{R}_{\geq 0}$, the number of configurations of every timed automaton is infinite and therefore the set $\mathcal{S}$ is also infinite. Also, note that, because of the delay transitions defined above, from every (q, v) in $\mathcal{S}$ there are infinitely many outgoing delay transitions.

Remark 3. Every sequence of transitions can be transformed into an *equivalent* sequence where the delay and action transitions alternate. This is because of two reasons: (i) every consecutive delay transitions $(q, v) \to^{\delta_1} (q, v_1)$ and $(q, v_1) \to^{\delta_2} (q, v_2)$ can be replaced by the transition $(q, v) \to^{\delta_1+\delta_2} (q, v_2)$ and (ii) every consecutive action transitions can be modified by adding a zero delay transition in between, that is, $(q, v) \to^{t_1} (q_1, v_1) \to^{t_2} (q_2, v_2)$ is same as $(q, v) \to^{t_1} (q_1, v_1) \to^{0} (q_1, v_1 + 0) \to^{t_2} (q_2, v_2)$. Henceforth, it will be assumed that, *in every sequence of transitions, delay and action transitions alternate.*

Two consecutive delay and action transitions $(q, v) \to^{\delta} (q, v + \delta) \to^{t} (q', v')$, will sometimes be concisely denoted as $(q, v) \to^{\delta,t} (q', v')$.

Runs. A *run* of a timed automaton $\mathcal{A} = (Q, X, q_0, T, F)$ is a sequence of (delay and action) transitions starting from the initial configuration $(q_0, \mathbf{0})$:

$$(q_0, \mathbf{0}) \to^{\delta_1,t_1} (q_1, v_1) \to^{\delta_2,t_2} \cdots \to^{\delta_n,t_n} (q_n, v_n)$$

A run of $\mathcal{A}$ is called an *accepting run* if it ends in a configuration that contains an accepting state of $\mathcal{A}$, that is, if $q_n \in F$.

Example 2.1.2. Following is a run of the automaton $\mathcal{A}_1$ (depicted in Figure 2.1):

$$(q_0, (0, 0)) \to^{1,t_1} (q_0, (0, 1)) \to^{1,t_1} (q_0, (0, 2))$$

Both of the transitions in this run consist first of a delay of 1 time unit and then taking the self loop at q_0, which sets the value of the clock x back to 0. Note that, this is not an accepting run.

Example 2.1.3. The following is also a run of the automaton in Figure 2.1:

$$(q_0, (0, 0)) \to^{1,t_1} (q_0, (0, 1)) \to^{1,t_1} \cdots \to^{1,t_1} (q_0, (0, 10)) \to^{t_2} (q_1, (0, 10))$$

Every transition (except the last) is taken after a delay of 1 time unit and then taking the self loop at the state q_0. The last transition is due to the transition t_2 present in the automaton. Since the final configuration in this run contains the state q_1 and it is an accepting state, this is an accepting run.

Remark 4. For modelling convenience, sometimes, the states in Timed Automata are equipped with *invariants*. An invariant in a state is a constraint and a timed automaton can stay at a state only while the values of the clocks satisfy the invariant of the state. Also, a transition can be taken by the automaton if the resulting values of the clocks satisfy the invariant of the target state. However, as far as the reachability problem is concerned, the invariants at states can be shifted to the guards of suitable transitions. Therefore, for the sake of simplicity, invariants have not been considered in the syntax.

2.2 Updatable Timed Automata

Bouyer et al. introduced the model of Updatable Timed Automata (from now on, sometimes to be referred to as UTA) in [BDFP00a, BDFP00b]. This model extends the model of Timed Automata, by replacing resets with updates.

Updates generalize resets by allowing the values of clocks to be set to other values instead of only to 0. An update to a clock (written as up_x) maps the value of x to either a non-negative integer c, or to the value of a clock y (possibly x itself) or even to the sum $y+d$ (only when this value is non-negative), where y is a clock and d is an integer [2]. Given a set of clocks X, an update up maps the value of every clock x in X to the value of its update up_x. Therefore, an update up can be viewed as the collection $\{up_x \mid x \in X\}$. Note that, not updating a clock x can be expressed as the update $x := x$. Therefore, henceforth, it will be assumed that every update updates every clock. Given a set of clocks X, $\mathbb{U}(X)$ will be used to denote the set of all possible updates over X.

Given a valuation v and a clock $x \in X$, $up(v)$ assigns a value to x as follows:

$$up(v)(x) = \begin{cases} d & \text{if } up_x = d \\ v(x) & \text{if } up_x = x \\ v(y) + d & \text{if } up_x = y + d \end{cases}$$

Note that, updates are only *partial* functions. Given a valuation v and an update up, $up(v)$ need not be a valuation (as defined on Page 13). For example, consider the valuation v where $v(x) = 2$, $v(y) = 3$ and an update up such that $x := y - 4 \in up$. Then, $up(v)$ is not a valuation, since $up(v)(x) = 3 - 4 = -1 < 0$.

The syntax of Updatable Timed Automata is same as it is for Timed Automata (Definition 2.1) with transitions having updates in place of resets.

Definition 2.3 (Updatable Timed Automata [BDFP04])**.** *An updatable timed automaton $\mathcal{A}$ is a tuple (Q, X, q_0, T, F) where Q is a finite set of states, X is a finite set of clocks, q_0 is the initial state, $T \subseteq Q \times \Phi(X) \times \mathbb{U}(X) \times Q$ is a finite set of transitions and $F \subseteq Q$ is the set of all accepting states.*

[2] In the UTA model introduced by Bouyer et al. the authors also consider *non-deterministic updates*, that are expressions of the form $x :\sim y + d$ where $\sim \in \{<, \leq, \neq, \geq, >\}$. However, this thesis does not consider these updates and only considers the so-called *deterministic updates*.

Example 2.2.1. An updatable timed automaton $\mathcal{A}_2 = (Q, X, q_0, T, F)$ is depicted in Figure 2.2, with $Q = \{q_0, q_1\}$, the set of all clocks $X = \{x, y\}$, two transitions $t_1 = (q_0,\ x = 1,\ y := x + 10,\ q_0)$ and $t_2 = (q_0,\ y - x = 10,\ \{\},\ q_1)$ and $F = \{q_1\}$.

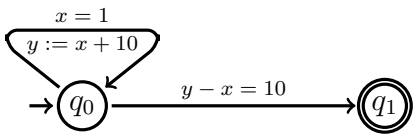


Figure 2.2: Example updatable timed automaton $\mathcal{A}_2$

Remark 5. Updates of the form $x := x$ are not depicted while drawing Updatable Timed Automata.

The semantics of Updatable Timed Automata is also same as that of Timed Automata (Definition 2.2) with only the action transitions being modified.

Definition 2.4 (Semantics of Updatable Timed Automata)**.** *The semantics of an updatable timed automaton $\mathcal{A} = (Q, X, q_0, T, F)$ is specified using a transition system $\mathcal{S}_\mathcal{A} = (\mathcal{S}, E)$ where $\mathcal{S}$ is the set of configurations (as defined on Page 13) of $\mathcal{A}$. The transitions present in E are of two types:*

delay: *for every configuration (q, v) of $\mathcal{A}$ and every constant $\delta \in \mathbb{R}_{\geq 0}$ there is a* delay *transition $(q, v) \to^\delta (q, v + \delta)$,*

action: *for every transition $t = (q, g, up, q') \in T$ and every valuation v such that (i) $v \models g$ and (ii) $up(v)$ is a valuation, there is an* action *transition $(q, v) \to^t (q', v')$, where $v' = up(v)$.*

Example 2.2.2. The following is an accepting run of the updatable timed automaton depicted in Figure 2.2:

$$(q_0, (0, 0)) \to^1 (q_0, (1, 1)) \to^{t_1} (q_0, (1, 11)) \to^{t_2} (q_1, (1, 11))$$

The first transition in this run is a delay transition of 1 time unit. The next transition is due to the self loop at q_0. This transition could be taken because the value of the clock x is 1. Now, this transition sets the value of the clock y to the value of x plus 10, that is, it sets the value of y to 11. At this point the guard in the transition $q_0 \to q_1$ is satisfied and this is the final transition present in the run.

Remark 6. Timed Automata is a subclass of Updatable Timed Automata where updates are restricted to be expressions only of the form $x := 0$ or $x := x$, with x being a clock. This can be observed from the fact that the syntax (Definition 2.3) and semantics (Definition 2.4) of Updatable Timed Automata extend the syntax (Definition 2.1) and semantics (Definition 2.2) of Timed Automata.

2.3 The Reachability problem

This thesis deals with the problem of checking reachability. Given an updatable timed automaton $\mathcal{A} = (Q, X, q_0, T, F)$, a state $q \in Q$ is said to be *reachable*, if there exists a run of $\mathcal{A}$ that ends in a configuration (q, v), for some valuation $v \in \mathbb{R}^X_{\geq 0}$. The remaining part of this chapter recalls the general framework of solving reachability in the restricted class of diagonal-free Timed Automata, that is used by the tools like UPPAAL [LPY97] and TChecker [HP]. The state-of-the-art algorithms for checking reachability in Timed Automata (when it contain diagonal constraints) and in Updatable Timed Automata are also recalled in this chapter.

The reachability problem. Given an updatable timed automaton $\mathcal{A}$ and a state q of $\mathcal{A}$, the reachability problem asks: is q reachable in $\mathcal{A}$?

The objective of this problem is therefore finding a configuration (q, v), with some valuation v, that is reachable from the initial configuration $(q_0, \mathbf{0})$. The challenge is, since valuations map clocks to non-negative real numbers, the set of all configurations (and hence all possible runs) of the automaton $\mathcal{A}$ is infinite. Therefore, in order to find if a configuration (q, v) is reachable in $\mathcal{A}$, it is not feasible to enumerate all configurations (or all possible runs) of $\mathcal{A}$.

However, in order to conclude that a state q is unreachable in an automaton $\mathcal{A}$, it is indeed necessary to know the entire *reachable* set of configurations of $\mathcal{A}$. In [AD94], Alur and Dill proved that checking reachability is decidable in Timed Automata. Moreover, it was proved that this problem is PSPACE-complete.

Theorem 2.5 ([AD94])**.** *Given a timed automaton $\mathcal{A}$ and a state q, checking if q is reachable in $\mathcal{A}$ is* PSPACE*-complete.*

Bouyer et al. showed in [BDFP04] that reachability becomes undecidable when considering the model of Updatable Timed Automata (Definition 2.3).

Theorem 2.6 ([BDFP04])**.** *Given an updatable timed automaton $\mathcal{A}$ and a state q, checking if q is reachable in $\mathcal{A}$ is undecidable.*

However, the authors also listed down some of the subclasses of Updatable Timed Automata for which checking reachability is decidable and also, in fact, PSPACE-complete. The crux in deciding reachability lies in finding a finite representation of $\mathcal{S}_\mathcal{A}$ (Definitions 2.2 and 2.4). The number of states in the automaton $\mathcal{A}$ is finite, thus the infiniteness of $\mathcal{S}$ is in fact due to the number of valuations being infinite. Therefore, to find a finite representation of $\mathcal{S}$, it is necessary to work with collections of valuations. Two kinds of collections of valuations have been proposed in the literature – regions and zones, these are discussed in the next two sections.

Remark 7. In several works, including [AD94, BDFP04], the automata models have been studied from a language theory perspective. The transitions of the automata contain *letter*s. With each run of the automaton, a *timed word* can be associated by concatenating the pairs consisting of the letters on the transitions

and the timestamps of each transition present in the run. An updatable timed automaton *accepts* a timed word if this word corresponds to an accepting run of the automaton. The *timed language* of a timed automaton is the set of all timed words accepted by it. The *emptiness* problem asks, given a timed automaton, whether its timed language is empty or not. The reachability problem corresponds to this emptiness problem. Given an automaton $\mathcal{A}$ and a state q, the reachability problem (asking if q is reachable in $\mathcal{A}$) is same as asking if the timed language of $\mathcal{A}$ (with q being the only *final* state) is non-empty. As far as the reachability problem is concerned, the letters on the transitions are not important and only the timestamps matter. This is the reason why labels on transitions have been ignored in the syntax.

2.4 Regions

Introduced by Alur and Dill in [AD94] for Timed Automata, *regions* are collections of "equivalent" valuations. Given a timed automaton, the regions corresponding to this automaton collect valuations that are *indistinguishable* by *all possible* guards of this automaton. Regions are the equivalence classes of an equivalence relation parameterized by the following "bound function".

For a set of clocks X, let $M\colon X \to \mathbb{N} \cup \{-\infty\}$ be a bound function mapping each clock in X to either a non-negative integer or $-\infty$. Given a clock $x \in X$, let M_x denote the constant associated to the clock x by the function M.

The following is a region equivalence, that gives rise to an algorithm for checking reachability for the class of diagonal-free Timed Automata.

Definition 2.7 (Region equivalence $\simeq_M$ [AD94])**.** *Given two valuations v, v' and a bounds function M, the relation $v \simeq_M v'$ holds if for every pair of clocks x, y in X –*

1. *$v(x) > M_x$ iff $v'(x) > M_x$,*
2. *if $v(x) \leq M_x$ then –*
   (a) *$\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$,*
   (b) *$\{v(x)\} = 0$ iff $\{v'(x)\} = 0$,*
   (c) *if $v(y) \leq M_y$ then $\{v(x)\} \leq \{v(y)\}$ iff $\{v'(x)\} \leq \{v'(y)\}$.*

In the definition above and the rest of the thesis, given a constant $c \in \mathbb{R}$, the notation $\lfloor c \rfloor$ denotes the integral part of c and $\{c\}$ denotes the fractional part of c. For example, for the constant 2.3, $\lfloor 2.3 \rfloor = 2$ and $\{2.3\} = 0.3$.

Every equivalence class of $\simeq_M$ is called an *M-region*. Note that, the regions are defined for every bounds function M. Let $\mathcal{A} = (Q, X, q_0, T, F)$ be a timed automaton without diagonal constraints. A bounds function (called the *maximal bounds*) $M_{\mathcal{A}}$ can be defined corresponding to this automaton $\mathcal{A}$ in the following manner: $M_{\mathcal{A}}$ maps every clock x to a constant $M_x \in \mathbb{N}$, where M_x is the maximum constant that is compared with the clock x, in a guard present in $\mathcal{A}$. If a clock x is not present in any of the guards of $\mathcal{A}$ then M_x is assigned to $-\infty$. The relation $\simeq_{M_{\mathcal{A}}}$ defined with respect to this function $M_{\mathcal{A}}$ is the region equivalence

corresponding to the automaton $\mathcal{A}$ and each equivalence class of $\simeq_{M_\mathcal{A}}$ is called a region corresponding to $\mathcal{A}$. For example, the regions corresponding to a timed automaton (without diagonal constraints) having two clocks $\{x, y\}$ and for which the maximal bound M is such that $M_x = 3$ and $M_y = 2$, are depicted in Figure 2.3.

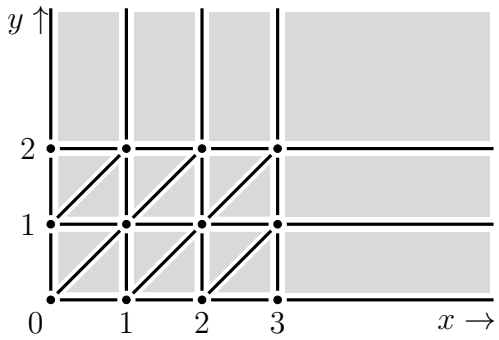


Figure 2.3: Regions corresponding to every diagonal-free timed automata with $M_x = 3$ and $M_y = 2$: each dot, each line segment and each shaded triangle and rectangle are the individual regions

Note that, for every timed automata having the same maximal bounds, the set of all regions are the same as well.

Region Automaton. Let $\mathcal{A} = (Q, X, q_0, T, F)$ be a timed automaton without diagonal constraints. Given $\mathcal{A}$, the regions corresponding to $\mathcal{A}$ (as shown in Figure 2.3) can be constructed. With these regions, a new finite-state automaton $\mathcal{R}(\mathcal{A})$ can be defined whose states are of the form $(q, \mathcal{R})$ where q is a state of the automaton $\mathcal{A}$ and $\mathcal{R}$ is a region of $\mathcal{A}$. The region automaton contains a transition $(q_1, \mathcal{R}_1) \to^t (q_2, \mathcal{R}_2)$ if there exists two valuations $v_1 \in \mathcal{R}_1$ and $v_2 \in \mathcal{R}_2$, such that $(q_1, v_1) \to^{\delta,t} (q_2, v_2)$ is possible in $\mathcal{A}$ for some $\delta \in \mathbb{R}_{\geq 0}$ and some transition t of $\mathcal{A}$. The initial state of this automaton is $(q_0, [\mathbf{0}])$ where $[\mathbf{0}]$ denotes the region containing the initial valuation $\mathbf{0} : X \to \{0\}$. Every state $(q, \mathcal{R})$ where $q \in F$ is an accepting state of this automaton $\mathcal{R}(\mathcal{A})$. Since the number of regions and the number of states of $\mathcal{A}$ are finite, the number of states of $\mathcal{R}(\mathcal{A})$ is also finite. Therefore, the set of all runs in $\mathcal{R}(\mathcal{A})$ is finite and this captures the (uncountably infinite) set of all possible runs of $\mathcal{A}$, thus providing a finite abstraction of $\mathcal{S}_\mathcal{A}$ (Definition 2.2).

This provides an algorithm for checking reachability in diagonal-free Timed Automata. Given such an automaton $\mathcal{A}$ and a state q, construct the automaton $\mathcal{R}(\mathcal{A})$ and check if (q, R) is reachable for some region R. Since $\mathcal{R}(\mathcal{A})$ is a finite state automaton, this can be checked.

The equivalence relation in Definition 2.7 however fails when the underlying automaton contains diagonal constraints. For example, let $\mathcal{A}$ be a timed automaton with $M_x = 3$ and $M_y = 2$ and assume $\mathcal{A}$ contains a guard $x - y \leq 2$. According to the region construction depicted in Figure 2.3, the following two valuations belong to the same region:

$$v_1 = \begin{cases} x \mapsto 3.5 \\ y \mapsto 1.8 \end{cases} \quad \text{and} \quad v_2 = \begin{cases} x \mapsto 3.5 \\ y \mapsto 1.2 \end{cases}$$

However, note that $v_1 \models x - y \leq 2$ but $v_2 \not\models x - y \leq 2$. Therefore, although the two valuations equisatisfy every non-diagonal constraint with constants lesser than the maximal constant, these do not equisatisfy every diagonal constraint, in particular, the diagonal $x - y \leq 2$. Therefore, a finer partitioning is required when the underlying automaton contains diagonal constraints. Bengtsson and Yi described this partitioning in [BY03]. Given a timed automaton (containing diagonal constraints) the set of all regions consists of all the regions for diagonal-free Timed Automata (Definition 2.7), with some of them being further divided based on each of the diagonal constraints present in the automaton.

Definition 2.8 (Region equivalence $\simeq_M^d$ [BY03])**.** *Given two valuations v, v', a bound function M and a set of diagonal constraints G, the relation $v \simeq_M^d v'$ holds if the following two conditions hold:*

1. *$v \simeq_M v'$ (Definition 2.7),*
2. *for every $\varphi \in G$, $v \models \varphi$ iff $v' \models \varphi$.*

To give an example of this modified region construction, consider a timed automaton $\mathcal{A}$ with $M_x = 3$ and $M_y = 2$. Additionally, assume that $\mathcal{A}$ contains two diagonal constraints $x - y \leq 2$ and $y - x \leq 1$. The regions corresponding to $\mathcal{A}$, based on the equivalence relation in Definition 2.8, are depicted in Figure 2.4.

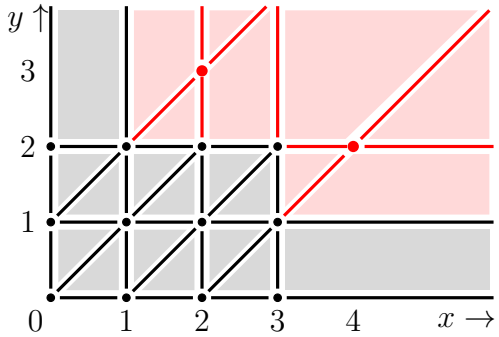


Figure 2.4: Regions corresponding to every timed automata with $M_x = 3$, $M_y = 2$ and two diagonal constraints $x - y \leq 2$, $y - x \leq 1$ present in some guard: each dot, each line segment and each shaded triangle and rectangle are the individual regions; the dots that are connected with line segments together form the region

Given a timed automaton $\mathcal{A}$, a region automaton similar to the one discussed on Page 19 can be constructed with respect to this finer equivalence relation $\simeq_M^d$. Call this region automaton $\mathcal{R}^d(\mathcal{A})$. Similar to the diagonal-free case, this can again be used to check reachability. Given a timed automaton $\mathcal{A}$, now containing diagonal constraints, whether a state q is reachable, can be checked by first constructing $\mathcal{R}^d(\mathcal{A})$ and then checking if (q, R) is reachable for some region R.

Regions can be further modified to take into account some updates in the underlying automaton. Bouyer et al. provided region constructions for some subclasses of Updatable Timed Automata in [BDFP04], while proving that reachability is decidable in those classes. Although regions provide a finite representation of the

transition system $\mathcal{S}_{\mathcal{A}}$ (Definition 2.2) and therefore an algorithm for deciding reachability, the downside of using regions is that the size of the region automaton is exponential in the number of clocks present in $\mathcal{A}$. This makes using regions impractical for checking reachability. However, regions are an important tool to provide theoretical guarantees. A constructive definition of the regions with respect to the equivalence relation in Definition 2.8, defined for timed automata when diagonal constraints are present, will be discussed and subsequently used in Chapter 4.

2.5 Zones

Like regions, *zones* are also collections of valuations. However, unlike regions, zones do not only collect "equivalent" valuations and therefore can be larger than regions. Zones are defined by a set of atomic constraints.

Definition 2.9 (Zone [DT98])**.** *A zone is a set of valuations, each of which satisfy a set of atomic constraints, that is, constraints of the form* $x \lhd c$, $c \lhd x$, $x - y \lhd c$ *or* $c \lhd x - y$, *where* x, y *are clocks,* $c \in \mathbb{N}$ *and* $\lhd \in \{<, \leq\}$.

Instead of viewing a zone as a set of valuations, zones can also be viewed as the set of atomic constraints defining it. Every valuation that satisfies all of the atomic constraints representing the zone, belongs to the zone.

Remark 8. The definition of a zone can be rephrased as follows: a zone Z is the intersection of the sets $[\![\varphi]\!]$ (this notation has been defined on Page 13) for every constraint φ defining the zone Z, that is, Z is the set $\bigcap_{\varphi \in Z} [\![\varphi]\!]$. From this definition, it can be assumed that, zones do not contain redundant constraints. This means, no zone contains two different constraints $x \lhd_1 c_1$ and $x \lhd_2 c_2$. This is because one of these two constraints must imply the other, that is, $[\![x \lhd_1 c_1]\!] \cap [\![x \lhd_2 c_2]\!]$ is either $[\![x \lhd_1 c_1]\!]$ or $[\![x \lhd_2 c_2]\!]$, therefore it is sufficient to only keep one of these two constraints in a zone. Same argument holds for the atomic constraints of the form $c \lhd x$, $x - y \lhd c$ and $c \lhd x - y$.

For notational convenience, a special constant x_0, called the *zero clock*, is often used. This constant is called a clock, albeit its value remains 0 forever. For every valuation $v \in \mathbb{R}^X_{\geq 0}$, $v(x_0) = 0$, assuming $x_0 \in X$, and for every delay $\delta \in \mathbb{R}_{\geq 0}$, $v(x_0) + \delta = 0$. Zero clock helps write non-diagonal constraints also as diagonal constraints, for example, the constraint $x \lhd c$ can be written as $x - x_0 \lhd c$. With the help of zero clock and the above remark in place, it will now be assumed that every zone Z contains only a single constraint $y - x \lhd_{xy} z_{xy}$ for every pair of clocks x, y where either x or y can also be the zero clock. Given a zone Z, $Z_{xy} = (\lhd_{xy}, z_{xy})$ represents the constraint $y - x \lhd_{xy} z_{xy}$ present in the zone Z, where x and y are two distinct clocks (can be the zero clock), $\lhd_{xy} \in \{<, \leq\}$ and $z_{xy} \in \mathbb{Z}$. Note that, Z_{xy} denotes the atomic constraint involving the difference $y - x$ and not $x - y$. This convention is chosen to align with a graph representation of zones described later.

Throughout this thesis, the following arithmetic will be used over the pairs $(\lhd, z)$ where $\lhd \in \{<, \leq\}$ and $z \in \mathbb{Z}$: $(\leq, z_1) + (\leq, z_2) = (\leq, z_1 + z_2)$ and $(<, z_1) + (<, z_2) = (<, z_1) + (\leq, z_2) = (\leq, z_1) + (<, z_2) = (<, z_1 + z_2)$. Also, given two such pairs $(\lhd_1, z_1)$

and $(\lhd_2, z_2)$, the inequality $(\lhd_1, z_1) < (\lhd_2, z_2)$ holds if either $z_1 < z_2$, or, $z_1 = z_2$ and $\lhd_1 = <$ and $\lhd_2 = \leq$. Whereas, $(\lhd_1, z_1) = (\lhd_2, z_2)$ if $z_1 = z_2$ and $\lhd_1 = \lhd_2$.

Definition 2.10 (Canonical Zone)**.** *A zone Z is said to be canonical if for every three distinct clocks x, y and w, with one of these three clocks allowed to be the zero clock, the inequality $Z_{xy} \leq Z_{xw} + Z_{wy}$ holds.*

Canonical zones enjoy an important property: for every constraint describing such a zone, there exists a valuation in the zone for which the bound on the constraint is "tight". The following proposition describes this property formally. This proposition will be useful in a number of proofs that are part of Chapter 4.

Proposition 2.11. *Given a non-empty canonical zone Z and two clocks x, y (one of these clocks can be the zero clock), if Z_{xy} contains the weak inequality, that is, if Z_{xy} is the pair $(\leq, z_{xy})$ then Z contains a valuation v such that $v(y) - v(x) = z_{xy}$. Whereas, if Z_{xy} contains the strict inequality, that is, if Z_{xy} is the pair $(<, z_{xy})$ then for every $\varepsilon \in \mathbb{R}_{>0}$, there exists $v \in Z$ such that $z_{xy} - \varepsilon < v(y) - v(x) < z_{xy}$.*

Distance graph.

A *distance graph* is a graph representing a set of atomic constraints. The vertices of a distance graph represent clocks and each edge represents a constraint. A special vertex 0 is used to represent the zero clock. The edge in Figure 2.5 encodes the constraint $x - y \lhd c$, where x, y are clocks with y being allowed to be the zero clock.

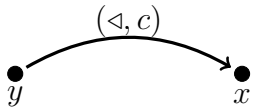


Figure 2.5: Representation of the constraint $x - y \lhd c$

The lower bound constraints are not represented as is. Consider the constraint $d \lhd x - y$, with y allowed to be zero clock. This constraint can be rewritten as $y - x \lhd -d$. This rewritten constraint is encoded using the edge in Figure 2.6.

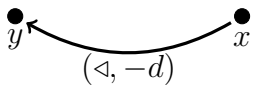


Figure 2.6: Representation of the constraint $d \lhd x - y$, that is, $y - x \lhd -d$

For a distance graph G, the notation $[\![\mathsf{G}]\!]$ will be used to denote the set of all valuations that satisfy each of the constraints represented by G. Note that, every set of atomic constraints has a unique distance graph representation.

Distance graphs provide a useful method of arguing whether a set of atomic constraints is satisfiable or not. This is presented in the following lemma.

Lemma 2.12 ([HSW16])**.** *Let G be a distance graph representing a set of atomic constraints over some set of clocks. Then, $[\![\mathsf{G}]\!]$ is empty iff G contains a negative cycle.*

Let $\mathsf{G}_1 = (V, E_1)$ and $\mathsf{G}_2 = (V, E_2)$ be two distance graphs representing two sets of atomic constraints over the same set of clocks. Therefore, the vertex sets of G_1 and G_2 are the same set V. Given such $\mathsf{G}_1, \mathsf{G}_2$, define the graph $\mathsf{min}(\mathsf{G}_1, \mathsf{G}_2) = (V, E)$ where E contains an edge $x \xrightarrow{\min(w_1,w_2)} y$, where w_1 is the weight of the edge $x \to y$ in G_1 and w_2 is the weight of the same edge in the graph G_2.

Lemma 2.13. *Let G_1 and G_2 be two graphs representing two sets of atomic constraints over the same set of clocks. Then, $[\![\mathsf{min}(\mathsf{G}_1, \mathsf{G}_2)]\!] = [\![\mathsf{G}_1]\!] \cap [\![\mathsf{G}_2]\!]$.*

Since zones are finite sets of atomic constraints, zones can also be represented as distance graphs. This representation of zones, along with the two results mentioned for distance graphs, will be used in Chapter 4.

2.6 Zone based reachability algorithm

Since the region automaton is exponential in size, it is not used in practice to check reachability. Instead, a zone based algorithm is used. This algorithm builds a graph, called the *zone graph*, whose nodes are pairs of the form (q, Z) where q is a state of the given automaton and Z is a zone. Once the required state appears in a node, the algorithm declares the state to be reachable. If no such node appears in the entire graph, the algorithm concludes that the state is unreachable. This section recalls this standard zone based reachability algorithm for *diagonal-free* Timed Automata, used by tools including UPPAAL [LPY97], KRONOS [Yov97] and TChecker [HP].

2.6.1 Zone Graph

Given a diagonal-free timed automaton $\mathcal{A}$, the reachability algorithm builds the zone graph corresponding to $\mathcal{A}$. The vertices of the zone graph are pairs of the form (q, Z) where q is a state of $\mathcal{A}$ and Z is a zone. The vertices of zone graph are called *nodes*. The zone graph contains a node (q_0, Z_0), where q_0 is the initial state of $\mathcal{A}$ and Z_0 is the following zone:

$$Z_0 := \left(\bigcup_{x \in X} \{0 \leq x\} \right) \cup \left(\bigcup_{x \in X, y \in X, x \neq y} \{x - y \leq 0\} \right) \tag{2.3}$$

This initial zone Z_0 contains all the valuations that can be written as $\mathbf{0} + \delta$ for some $\delta \in \mathbb{R}_{\geq 0}$, where $\mathbf{0}$ is the *zero valuation* mapping every clock to 0. For every transition (q, g, R, q') of $\mathcal{A}$ and every node (q, Z) present in the zone graph, there exists an edge from (q, Z) to its *successor* (q', Z'), where Z' consists of the valuations v' such that there exists a valuation $v \in Z$ with $v \models g$ and $v' = [R]v + \delta$, for some $\delta \in \mathbb{R}_{\geq 0}$. The zone Z' is computed according to the following expression:

$$Z' = \overrightarrow{[R](Z \wedge g)} \tag{2.4}$$

On the above expression, $Z \wedge g$ is the zone consisting of all the valuations from Z that satisfy the guard g. If Z is viewed as the set of atomic constraints defining

it, then $Z \wedge g$ is the set consisting of the atomic constraints defining Z and the atomic constraints present in g. Computing this successor of a node in the zone graph requires the following three operations to be performed on zones.

Intersection with a constraint. Given a zone Z and a constraint φ, $Z \wedge \varphi$ is the set $\{v' \mid v' \in Z \text{ and } v' \models \varphi\}$.

Reset. Given a zone Z and a set of clocks R, the reset of Z with respect to R, written as $[R](Z)$, is the set $\{v' \mid v' = [R](v) \text{ for some } v \in Z\}$.

Time elapse. Given a zone Z, the time elapse operation produces the set of valuations $\overrightarrow{Z} = \{v' \mid v' = v + \delta \text{ for some } v \in Z \text{ and } \delta \in \mathbb{R}_{\geq 0}\}$. A zone Z is called *time-elapsed* if $Z = \overrightarrow{Z}$, in other words, for every valuation $v \in Z$ and every delay $\delta \in \mathbb{R}_{\geq 0}$, the valuation $v + \delta \in Z$.

Theorem 2.14 ([DT98])**.** *Given a zone Z, a constraint φ and a set of clocks R, each of the sets of valuations $Z \wedge \varphi$, $[R](Z)$ and $\overrightarrow{Z}$ are zones as well.*

Instead of Timed Automata, when considering Updatable Timed Automata, the successor of a node (q, Z) in the zone graph with respect to a transition (q, g, up, q') can be computed according to the following expression:

$$Z' = \overrightarrow{up(Z \wedge g)} \tag{2.5}$$

Starting from the initial node (q_0, Z_0), the zone graph can be built by repeatedly computing successors (according to 2.4 or 2.5, depending on whether resets or updates are present). However, this does not immediately produce an algorithm, since this procedure is not guaranteed to terminate. For example, consider the automaton $\mathcal{A}$ in Figure 2.7. The zone graph of this automaton is depicted in Figure 2.8 and is infinite. The self loop in the state q_1 is the reason behind this infiniteness.

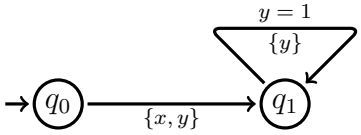


Figure 2.7: Timed automaton $\mathcal{A}$ with infinite zone graph

It turns out, as far as the reachability problem is concerned, there can be nodes in the zone graph for which no successors need to be computed. Stopping explorations from such nodes ensure that exploring only a finite portion of the zone graph is sufficient for checking reachability. Two procedures exist for finding out (sufficiently many) such nodes in the zone graphs of diagonal-free timed automata. These are described in the next two sections.

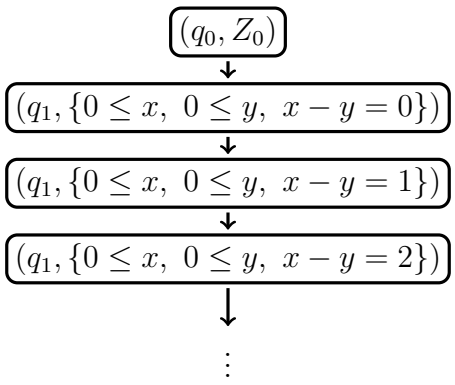


Figure 2.8: Zone graph of the automaton $\mathcal{A}$ in Figure 2.7; the constraints $x - y = i$ in the zones are actually the two constraints $x - y \leq i$ and $i \leq x - y$, $i \in \mathbb{N}$

2.6.2 Building the zone graph

Given a diagonal-free timed automaton $\mathcal{A} = (Q, X, q_0, T, F)$, the reachability algorithm builds the zone graph of $\mathcal{A}$, starting with the initial node (q_0, Z_0) where q_0 is the initial state of $\mathcal{A}$ and Z_0 is the zone described in (2.3), and then by computing the successors (according to (2.4)) of the nodes present in the graph with respect to the transitions of the underlying automaton $\mathcal{A}$. Assuming q is the state of $\mathcal{A}$ for which reachability is being checked, if q appears in a node of the zone graph then q is indeed reachable in $\mathcal{A}$, and conversely, if q is reachable in $\mathcal{A}$, then q appears in some node of the zone graph of $\mathcal{A}$.

Theorem 2.15 ([DT98]). *Zone graphs, even if infinite, are sound and complete for control-state reachability.*

While building the zone graph, a node is said to be *discovered*, when it becomes the successor of some existing node in the graph. A node is termed *explored* when each of its successors are computed. As illustrated in the previous section, like for the automaton in Figure 2.7, the zone graphs are not always finite and therefore continuous exploration of existing nodes may not terminate in general. Therefore, a method is required to "safely truncate" the zone graph by stopping redundant explorations. By "safely" it is meant that, if a state of the automaton $\mathcal{A}$ appears in the original (possibly infinite) zone graph, then it should also appear in the truncated zone graph and vice versa. This is achieved by using *subsumption relations* between nodes. A state q' is said to be *reachable* from a node (q, Z), if there exists a path from (q, Z) in the zone graph reaching a node (q', Z') with some zone Z'. A subsumption relation is a relation that ensures the following: if (q, Z'') is subsumed by (q, Z) then every state that is *reachable* from (q, Z'') is also *reachable* from (q, Z). This implies that, whenever (q, Z'') is subsumed by (q, Z), exploring (q, Z'') is redundant when (q, Z) is already explored or is going to be explored. Note that, a subsumption relation can only be defined between two nodes having the same state.

A pseudocode for the reachability algorithm with the optimization of not exploring nodes that get subsumed by some other node, is described in Algorithm 1. Every valid subsumption relation can be plugged in Line 11 in place of is_subsumed_by.

Input: $\mathcal{A} = (Q, X, q_0, T, F)$
Output: *Yes*: if some state in F is reachable; *No*: otherwise

```
1  passed = ∅, waiting = ∅ ;
2  waiting ← waiting ∪ {(q0, Z0)} ;
3  while waiting ≠ ∅ do
4  |  (q, Z) ← pop(waiting) ;
5  |  passed ← passed ∪ {(q, Z)} ;
6  |  forall outgoing transitions t = (q, g, R, q') from q do
7  |  |  (q', Z') ← next((q, Z), t) ;
8  |  |  if q' ∈ F then
9  |  |  |  return Yes ;
10 |  |  end
11 |  |  if ∃(q', Z'') ∈ passed ∪ waiting s.t. (q', Z') is_subsumed_by (q', Z'')
   |  |   then
12 |  |  |  go to the next iteration in line 5 ;
13 |  |  else
14 |  |  |  waiting ← waiting ∪ {(q', Z')} ;
15 |  |  end
16 |  end
17 end
18 return No ;
```

Algorithm 1: Zone graph enumeration with arbitrary subsumption relation

One simple subsumption relation is the inclusion relation ($\subseteq$), which can be defined between two nodes as: $(q, Z) \subseteq (q, Z')$ if $Z \subseteq Z'$. From this relation it follows that whenever $(q, Z) \subseteq (q, Z')$, every state reachable from (q, Z) is also reachable from (q, Z'), since Z' contains all the valuations present in Z. This inclusion relation, however, fails to ensure termination of Algorithm 1 for certain diagonal-free timed automata. For example, note that, this algorithm with the inclusion relation will not terminate for the automaton in Figure 2.7. This is because the infinitely many zones appearing in its zone graph (Figure 2.8) are all disjoint, $\{0 \leq x,\ 0 \leq y,\ x - y = i\} \cap \{0 \leq x,\ 0 \leq y,\ x - y = j\} = \emptyset$, for every $i, j \in \mathbb{N},\ i \neq j$.

Two stronger operators have been studied in the literature to prune redundant explorations, guaranteeing termination of Algorithm 1 for timed automata *without diagonal constraints*: (i) extrapolation and (ii) simulation relation. Since extrapolation operators do not work in the presence of diagonal constraints [Bou03], these are not discussed further. More details about extrapolation operators can be found in [DT98, BBLP06]. The rest of the thesis only considers simulation relations as the means for guaranteeing termination of the zone graph enumeration. Simulation relations can be plugged into Line 11 of Algorithm 1 in place of is_subsumed_by.

In Algorithm 1, the function next computes the successor of a node with respect to a transition according to 2.4 or 2.5 depending on whether in the input automaton contains only resets or updates. This function uses the three operations on zones discussed on Page 24. Also, every zone that appears in the zone graph

enumeration is *canonicalized*. In general, transforming a zone into a canonical zone (Definition 2.10) takes $\mathcal{O}(n^3)$ time, n being the number of clocks present in the zone. This canonicalization can be done using the Floyd-Warshall's shortest path algorithm [BY03, CLRS09].

The simulation relations (to be discussed in the next section) are parameterized by *clock bounds* functions. One such function the M bounds has been discussed on Page 18 while defining regions. Another bound function LU has been introduced in [BBLP06] distinguishing the lower and upper bound constraints present in the guards. More details about the LU bounds will be discussed in Chapter 3.

2.6.3 Simulation Relation

Simulation relations provide a method for reducing "redundant" explorations of the zone graph. Some simulation relations guarantee finiteness of the zone graph. Simulation relations ensure: if (q, Z) is simulated by (q, Z'), written as $(q, Z) \sqsubseteq (q, Z')$, then every state reachable from (q, Z) is also reachable from (q, Z').

Simulation relations are defined between configurations of Timed Automata (or Updatable Timed Automata, in general). These then naturally extend to a relation between two nodes. When a configuration (q, v) *is simulated by* (q, v'), written as $(q, v) \sqsubseteq (q, v')$, it implies that every sequence of transitions that are possible from (q, v) are also possible from (q, v'). Below is the definition of simulation relations.

Definition 2.16 (Simulation relation)**.** *Given an updatable timed automaton $\mathcal{A} = (Q, X, q_0, T, F)$, a reflexive and transitive relation $\sqsubseteq$ defined between two configurations having the same state, say (q, v_1) and (q, v_2), is called a* simulation relation *for $\mathcal{A}$, if it satisfies the following two conditions:*

1. *for every $\delta \geq 0$ if $(q, v_1) \sqsubseteq (q, v_2)$ then $(q, v_1 + \delta) \sqsubseteq (q, v_2 + \delta)$,*
2. *for every transition $t = (q, g, R, q')$ of $\mathcal{A}$, if $(q, v_1) \sqsubseteq (q, v_2)$ and if t is enabled from (q, v_1), that is, $(q, v_1) \rightarrow^t (q', [R](v_1))$ then firstly t is also enabled from (q, v_2), that is, $(q, v_2) \rightarrow^t (q', [R](v_2))$ and secondly $(q', [R](v_1)) \sqsubseteq (q', [R](v_2))$.*

The simulation relation defined above is actually called a *strong-time simulation relation*. Another kind of simulation relation has been defined in the literature, called *time-abstract simulation relation*. A time abstract simulation relation is a relation $\sqsubseteq_{t.a.}$ that follows the conditions of Definition 2.16, with the first condition being modified into: if the relation $(q, v_1) \sqsubseteq_{t.a.} (q, v_2)$ holds then for every $\delta \geq 0$ there exists some $\delta' \geq 0$ such that the relation $(q, v_1 + \delta) \sqsubseteq_{t.a.} (q, v_2 + \delta')$ holds as well. This thesis will only consider strong-time simulation relations and hence "simulation relation" will always refer to "strong-time simulation relation".

Simulation relations defined over configurations can be naturally lifted to relate nodes of the zone graph, in order for it to be applicable in a reachability algorithm. Given two nodes (q, Z) and (q, Z'), the first node is said to be "simulated by" the second, written as $(q, Z) \sqsubseteq (q, Z')$, if for every valuation $v \in Z$ there exists $v' \in Z'$ such that $(q, v) \sqsubseteq (q, v')$. Line 11 of Algorithm 1 can be modified in the following way to incorporate simulation relation as the subsumption relation:

$$(q', Z') \text{ is\_subsumed\_by } (q', Z'')$$
$$\downarrow$$
$$(q', Z') \sqsubseteq (q', Z'')$$
$$\downarrow$$
$$Z' \sqsubseteq Z''$$

The reachability algorithm (Algorithm 1) with a simulation relation as the subsumption relation is *sound* and *complete* for checking reachability. By *sound* it means that whenever the algorithm declares a state q to be reachable, the automaton contains a run leading upto q. By *complete* it means that if there exists a run of the automaton upto a state q, then the algorithm declares q to be reachable. However, to make sure the algorithm always terminates, an additional property is desired of simulation relations – *finiteness*. A simulation relation $\sqsubseteq$ is said to be *finite* if in every infinite sequence of zones $Z_1, Z_2, \ldots$, there exists two zones Z_i and Z_j with $i > j$ and $Z_i \sqsubseteq Z_j$.

Simulation relations for diagonal-free timed automata. Two finite simulation relations have been studied for timed automata without diagonal constraints, $\sqsubseteq_M$ [DT98, BBFL03] and $\sqsubseteq_{LU}$ [BBLP06]. Among these, $\sqsubseteq_{LU}$ is *coarser* than $\sqsubseteq_M$. This means, $\sqsubseteq_{LU}$ can result in more number of simulations than $\sqsubseteq_M$. Because of this, $\sqsubseteq_{LU}$ results in smaller zone graph computations and therefore faster reachability procedures. In practice, tools checking reachability for diagonal-free Timed Automata generally use the $\sqsubseteq_{LU}$ simulation relation over $\sqsubseteq_M$. More details about $\sqsubseteq_{LU}$ will be discussed in the upcoming Chapters 3 and 4.

2.6.4 Reachability algorithm: Overall framework

Given a diagonal-free timed automaton $\mathcal{A}$, the state-of-the-art reachability checking tools use Algorithm 1 with a simulation relation as the subsumption relation. Since the available simulation relations are all parameterized (by the so-called clock bounds M or LU), the overall reachability algorithm needs to compute these parameters first before starting the zone graph enumeration. This parameter computation is generally a static analysis on the input automaton. The M and LU bounds can be computed by solving a system of inequalities [BBFL03, BBLP06]. The overall reachability algorithm therefore consists of two steps:

Step 1. compute the parameters (static analysis)

Step 2. zone graph enumeration (Algorithm 1) with simulation

This thesis defines a simulation relation $\sqsubseteq_G$, parameterized by a finite set of atomic constraints G. Chapter 3 describes the static analysis to compute this parameter. Chapter 4 then describes an algorithm for checking $Z \sqsubseteq_G Z'$, given two zones Z, Z' and a set G. This can then be plugged into Algorithm 1 to get a procedure for checking reachability. This procedure terminates for every timed automaton (even with diagonal constraints) and for some of the classes (listed down in Chapter 5) of Updatable Timed Automata with decidable reachability.

Chapter 3

A new simulation relation

The "configuration space" of Timed Automata is uncountably infinite, due to the fact that the clocks range over non-negative real numbers. This makes checking reachability in Timed Automata an inherently hard task, to be precise, a PSPACE-hard (and complete) task [AD94]. In Updatable Timed Automata, the problem becomes even more difficult, in fact, undecidable [BDFP04]. However, interesting subclasses (retaining some of the updates) have been studied, where checking reachability remains decidable. These particular subclasses are not of interest in this chapter (some of these will be considered in Chapter 5). The results (to be presented) in this chapter consider the general classes of Timed Automata and Updatable Timed Automata. Therefore, these results also hold in those subclasses.

The infiniteness of the search space makes naïve searching infeasible to determine whether a state of the input automaton is reachable or not. The algorithms for checking such reachability rely on finite abstractions of the search space. The standard reachability algorithm builds a graph, called the *zone graph*. Each node of this graph is a pair consisting of a state of the input automaton and a *zone*. A state is declared to be reachable if it appears in a node in this graph. The challenge is that naïve exploration of this graph may not terminate. Removing redundant explorations – therefore guaranteeing termination of the reachability algorithm – is achieved by using *simulation* relations.

The class of Timed Automata *without* diagonal constraints is a subclass of Timed Automata that has been widely studied and has been used in practice, much more than the class containing diagonal constraints. There are perhaps two reasons for this: (i) diagonal constraints do not add any expressive power, and (ii) the availability of a (well-studied) simulation relation for diagonal-free Timed Automata – the LU simulation relation [BBLP06], which can also be checked efficiently [HSW16].

Diagonal constraints offer succinctness. Timed Automata with diagonal constraints can be up to exponentially more succinct than without diagonal constraints [BDGP98, BC05]. This therefore adds convenience while modelling a system. On the other hand, updates increase the expressivity. To the best of our knowledge, no simulation based reachability algorithm was known when Timed Automata contain

diagonal constraints and/or updates, prior to our work. The goal in this chapter is to propose a simulation relation that is correct for checking reachability in Timed Automata in the presence of diagonal constraints and updates.

The popular LU simulation relation is parameterized by two bounds functions L and U. Given an input automaton, each of these functions associates a non-negative integer (or $-\infty$) to every clock, based on the guards present in the transitions of the input automaton. But these maps have a potential drawback. These bounds are based only on the constants present in the automaton and are oblivious to the inequality present. For example, suppose the input automaton contains the constraint $x \leq 2$ in some guard, and no other constraint $x \leq c$ or $x < c$ with a larger c. Then the value of $U(x) = 2$. If the guard would have contained $x < 2$ instead, then also $U(x) = 2$. A couple of questions may arise at this point: (i) can a bounds function be defined that also considers the inequality and (ii) can that "little more precise" bound improve the simulation relation coming out of it?

This chapter defines the relation $\sqsubseteq_G$ (in Section 3.1). The set G in this relation consists of atomic constraints, thus, incorporating the inequality present in the guards as well. Now, does $\sqsubseteq_G$ improve upon the existing LU simulation? Since LU simulation is only defined for diagonal-free Timed Automata, when G contains only non-diagonal constraints, it turns out, $\sqsubseteq_G$ is coarser than LU (Section 3.2). However, the improvement is not drastic and therefore may not be visible in practice. Nevertheless, the relation $\sqsubseteq_G$ is also defined in the presence of diagonal constraints as well as updates, which indeed enables checking reachability in a class of timed automata larger than the class where the LU simulation can be used.

While using the LU simulation relation, instead of computing L, U for the entire input automaton, it is better to compute (possibly different) pairs of L, U bounds corresponding to each state of the input automaton [BBFL03]. The idea is to consider only the guards of the transitions that are "relevant" at each state to determine that state's L, U bounds. Section 3.3 incorporates this idea of state-specific bounds and defines a map $\mathcal{G}$, that associates a set of atomic constraints to each state of the input automaton, instead of considering a single G for the entire input automaton. The goal is to construct this map $\mathcal{G}$ so that when the relation $\sqsubseteq_G$ is defined with respect to this map $\mathcal{G}$, the resulting relation $\sqsubseteq_{\mathcal{G}}$ becomes a simulation relation. This section constructs this map, achieving this goal, in the presence of diagonal constraints and updates.

The state-specific L, U bounds are computed by solving a system of inequalities. Whereas, the map $\mathcal{G}$ will be determined by performing a fixpoint iteration. This iteration always terminates when the input automaton is a timed automaton. But, when the input is an updatable timed automaton, this fixpoint iteration is no longer guaranteed to terminate. Since the simulation relation $\sqsubseteq_{\mathcal{G}}$ can be used in a reachability algorithm only after computing the map $\mathcal{G}$, it is important to know, given an input automaton, whether the fixpoint iteration is going to terminate or not. This chapter concludes by providing a method for determining, given an input updatable timed automaton, whether this fixpoint iteration terminates or not (Section 3.4).

3.1 The relation $\sqsubseteq_G$

This section defines the relation $\sqsubseteq_G$, given a (finite or infinite) set of atomic constraints G. This relation is first defined between two valuations and subsequently between two zones. Upcoming Section 3.3 further extends these two relations to a relation between configurations and one between nodes, respectively. However, as will be shown, these two extended relations essentially boil down to the relations between valuations and zones, that are going to be defined in this section. Since simulation relations are relations defined between configurations, the aim in this chapter is to make the relation $\sqsubseteq_G$, defined between two configurations of a given updatable timed automaton, a simulation relation.

The definition of a simulation relation (Definition 2.16) is recalled below, this motivates the definition of $\sqsubseteq_G$. Given two configurations (q, v) and (q, v') of an updatable timed automaton $\mathcal{A}$, satisfying $(q, v) \sqsubseteq (q, v')$, the following three conditions need to hold in order for the relation $\sqsubseteq$ to be a simulation relation:

1. for every $\delta \in \mathbb{R}_{\geq 0}$, $(q, v + \delta) \sqsubseteq (q, v' + \delta)$,
2. for every transition $t = (q, g, up, q')$ of $\mathcal{A}$, if t is enabled from (q, v), then –
   (a) t is also enabled from (q, v'), and
   (b) $(q', up(v)) \sqsubseteq (q', up(v'))$.

To recall, a transition $t = (q, g, up, q')$ is said to be enabled from a configuration (q, v) if two conditions are met: (i) $v \models g$ and (ii) $up(v)$ is a valid valuation, that is, $up(v)(x) \geq 0$ for every clock x. The conditions 1 and 2a above motivate the following definition of the relation $\sqsubseteq_G$, defined between two valuations.

Definition 3.1 ($\sqsubseteq_G$ over valuations)**.** *Given two valuations v, v' and a (finite or infinite) set of atomic constraints G, $v \sqsubseteq_G v'$ if $\forall \varphi \in G$ and $\forall \delta \in \mathbb{R}_{\geq 0}$:*

$$\textit{if } \; v + \delta \models \varphi \; \textit{ then } \; v' + \delta \models \varphi \; .$$

Example 3.1.1. Let $G = \{x < 2\}$ and v and v' be two valuations such that $v(x) = 0.5$ and $v'(x) = 1$. Then $v \not\sqsubseteq_G v'$, since $v + 1.4 \models x < 2$ ($v(x) + 1.4 = 0.5 + 1.4 = 1.9 < 2$) but $v' + 1.4 \not\models x < 2$ ($v'(x) + 1.4 = 1 + 1.4 = 2.4 > 2$). However, $v' \sqsubseteq_G v$ holds, since $v(x) < v'(x)$, thus, for every $\delta \in \mathbb{R}_{\geq 0}$ if $(v' + \delta)(x) < 2$ then $(v + \delta)(x) < 2$ as well, in other words, $v' + \delta \models x < 2$ implies $v + \delta \models x < 2$.

Example 3.1.2. Let $G = \{x < 2,\ 0 \leq x - y\}$ and v, v' be two valuations such that $v(x) = 0.5,\ v(y) = 1$ and $v'(x) = 1,\ v'(y) = 0.2$. Then, $v \not\sqsubseteq_G v'$ since $v \not\sqsubseteq_{\{x<2\}} v'$ from Example 3.1.1. Moreover, $v' \not\sqsubseteq_G v$ since $v' \models 0 \leq x - y$ (since $v'(x) - v'(y) = 1 - 0.2 = 0.8 \geq 0$), but $v \not\models 0 \leq x - y$ (since $v(x) - v(y) = 0.5 - 1 = -0.5 \not\geq 0$).

Given a set of atomic constraints G, the next two propositions talk about the relations (of Definition 3.1) defined with respect to subsets of G. The following proposition states that if the relation $\sqsubseteq_G$ holds between two valuations for some set of atomic constraints G then the relation also holds for every set contained inside G. The proof follows directly from Definition 3.1.

Proposition 3.2. *Let G be a set of atomic constraints and v, v' be two valuations. Then, $v \sqsubseteq_G v'$ implies $v \sqsubseteq_{G'} v'$, for every G' satisfying $G \supseteq G'$.*

The reverse direction of the above proposition need not hold in general. However, the next proposition states that if the relations $v \sqsubseteq_{\{\varphi\}} v'$ hold *for every* $\varphi \in G$, then the relation $v \sqsubseteq_G v'$ holds as well.

Proposition 3.3. *Given a set of atomic constraints G and two valuations v, v', the relation $v \sqsubseteq_G v'$ holds iff for every $\varphi \in G$, the relation $v \sqsubseteq_{\{\varphi\}} v'$ holds.*

Proof. ($\Rightarrow$) This follows from Proposition 3.2 since $\{\varphi\} \subseteq G$ for every $\varphi \in G$.

($\Leftarrow$) Choose $\varphi \in G$ and $\delta \in \mathbb{R}_{\geq 0}$ such that $v + \delta \models \varphi$. Since $v \sqsubseteq_{\{\varphi\}} v'$, Definition 3.1 implies $v' + \delta \models \varphi$. Since φ was an arbitrary constraint from G, this implication holds for every $\varphi \in G$ and for every $\delta \in \mathbb{R}_{\geq 0}$, proving $v \sqsubseteq_G v'$. □

Example 3.1.1 illustrates that the relation in Definition 3.1 defined over valuations, is not symmetric. However, from Definition 3.1 it can be deduced easily that the relation $\sqsubseteq_G$ is reflexive and transitive, making it a preorder.

Since the reachability algorithms do not deal with valuations directly, and deal with zones instead, it is necessary to define a relation between a pair of zones. This can be achieved by lifting the relation in Definition 3.1 in the following manner.

Definition 3.4 ($\sqsubseteq_G$ over zones)**.** *Given two zones Z, Z' and a (finite or infinite) set of atomic constraints G, $Z \sqsubseteq_G Z'$ if $\forall v \in Z \;\; \exists v' \in Z'$ such that $v \sqsubseteq_G v'$.*

Note that, this relation (between zones) trivially holds when $Z = \emptyset$ and it does not hold if $Z \neq \emptyset$ but $Z' = \emptyset$. Zones contain infinitely many valuations, thus it is difficult to ascertain when the relation $\sqsubseteq_G$ holds between two non-empty zones and when it does not. Chapter 4 considers this problem and proposes an algorithm for checking $Z \sqsubseteq_G Z'$, when G is a given *finite* set of atomic constraints.

Section 3.3 extends the relation $\sqsubseteq_G$ defined between valuations (Definition 3.1) to a relation over configurations, that is, pairs of the form (q, v). Also, it extends the relation defined in Definition 3.4 (between two zones) to a relation between two nodes of the zone graph, that is, pairs of the form (q, Z). In both of these cases (configurations and nodes) the relations are defined between two pairs only when the states present in the pairs are the same. For this reason, the relations to be defined in Section 3.3 boil down to the relations defined in this section.

Now, before going to Section 3.3, the next section considers the restricted class of timed automata without diagonal constraints. Behrmann et al. in [BBLP06] proposed the state-of-the-art simulation relation $\sqsubseteq_{LU}$, for this class of automata. The next section compares $\sqsubseteq_{LU}$ with the relation $\sqsubseteq_G$ of Definition 3.1.

3.2 Comparing $\sqsubseteq_G$ with LU-preorder

The relation $\sqsubseteq_{LU}$ relating two valuations, proposed by Behrmann et al. in [BBLP06], is used in the state-of-the-art algorithms for checking reachability in diagonal-free

timed automata. The relation $\sqsubseteq_G$ defined between two valuations in the previous section (Definition 3.1), can also be defined for diagonal-free timed automata. The only restriction being, the set G will consist of only non-diagonal constraints. Now, with two relations being available for diagonal-free timed automata, a natural question arises: how do these two relations compare? This section answers this.

The relation $\sqsubseteq_{LU}$ is parameterized by two functions, L and U, together called an LU-bound. Given a diagonal-free timed automaton as input, these L, U bounds are computed based on the constants present in the guards of the automaton. On the other hand, the relation $\sqsubseteq_G$ is parameterized by a set of atomic constraints G. To emphasize the restriction on non-diagonal constraints, instead of using G, for the rest of this section, G^{nd} will be used to denote a set of non-diagonal constraints and $\sqsubseteq_{G^{nd}}$ will be the relation defined with respect to G^{nd} according to Definition 3.1.

Since L, U are defined based on the set of all guards present in the input automaton, these can also be defined based on arbitrary (finite) sets of atomic constraints. Theorem 3.13 will prove that $v \sqsubseteq_{G^{nd}} v'$ implies the relation $v \sqsubseteq_{LU} v'$, when the LU bounds are defined with respect to the set G^{nd} (Definition 3.12). The reverse direction does not hold. This proves that $\sqsubseteq_G$ relates more valuations than $\sqsubseteq_{LU}$, making the former coarser than the latter.

The aim now is to prove this coarseness, that is, the upcoming Theorem 3.13. Proposition 3.2 proved that $v \sqsubseteq_{G^{nd}} v'$ iff for every constraint $\varphi \in G^{nd}$, the relation $v \sqsubseteq_{\{\varphi\}} v'$ holds. Characterizing $v \sqsubseteq_{\{\varphi\}} v'$ will help prove Theorem 3.13. First, the relation $\sqsubseteq_{\{x \lhd c\}}$ (given a constraint $x \lhd c$) is characterized in the following theorem.

Theorem 3.5. *Given two valuations v, v' and a non-diagonal constraint $x \lhd c$ where $c \in \mathbb{N}$, the relation $v \sqsubseteq_{\{x \lhd c\}} v'$ holds iff either $v \not\models x \lhd c$ or $v'(x) \leq v(x)$.*

Proof. ($\Leftarrow$) If $v \not\models x \lhd c$ then clearly $\forall \delta \in \mathbb{R}_{\geq 0}\ v + \delta \not\models x \lhd c$ as well. Then the relation $v \sqsubseteq_{\{x \lhd c\}} v'$ holds trivially. Similarly, if $v'(x) \leq v(x)$ then for every $\delta \in \mathbb{R}_{\geq 0}$, $v'(x) + \delta \leq v(x) + \delta$ as well. Therefore, if $v(x) + \delta \lhd c$ then $v'(x) + \delta \lhd c$ also is true. This again implies the relation $v \sqsubseteq_{\{x \lhd c\}} v'$.

($\Rightarrow$) Conversely, let $v \sqsubseteq_{\{x \lhd c\}} v'$. There are two possibilities: (i) $v \not\models x \lhd c$ or (ii) $v \models x \lhd c$. In the second case, since $v \sqsubseteq_{\{x \lhd c\}} v'$, it follows that $v' \models x \lhd c$ as well, that is, $v'(x) \lhd c$. If $v'(x) > v(x)$ then from the density of real numbers, there exists $\delta \in \mathbb{R}_{\geq 0}$ such that $v(x) + \delta \lhd c$ however $v'(x) + \delta > c$. This contradicts the assumption that $v \sqsubseteq_{\{x \lhd c\}} v'$. Therefore, $v'(x) \leq v(x)$ must hold. □

The arithmetic defined over pairs of the form $(\lhd, c)$ on Page 21, where $\lhd \in \{<, \leq\}$ and $c \in \mathbb{Z}$, will be used in the proofs in this section.

The following result generalizes Theorem 3.5 from a single upper-bound non-diagonal to a set of upper-bound non-diagonal constraints. This result shows that, given a set of atomic constraints G^{nd}, as far as the relation $\sqsubseteq_{G^{nd}}$ is concerned, if G^{nd} contains more than one upper-bound constraint involving a clock x, then it is sufficient to keep only the upper-bound constraint involving x in G^{nd} that has the max pair $(\lhd, c)$, with respect to the order defined on Page 21.

Corollary 3.6. *Let $G^{nd} = \{x \lhd_1 c_1,\ \ldots\ ,\ x \lhd_k c_k\}$ be a finite set of non-diagonal upper-bound constraints involving the same clock x and v, v' be two valuations. Then, $v \sqsubseteq_{G^{nd}} v'$ iff $v \sqsubseteq_{\{x \lhd c\}} v'$, where $(\lhd, c) = \max\{(\lhd_i, c_i) \mid x \lhd_i c_i \in G^{nd}\}$.*

Proof. ($\Rightarrow$) Since $x \lhd c \in G^{nd}$, it follows from Proposition 3.3 that $v \sqsubseteq_{\{x \lhd c\}} v'$.

($\Leftarrow$) Assume the relation $v \sqsubseteq_{\{x \lhd c\}} v'$, where $(\lhd, c) = \max\{(\lhd_i, c_i) \mid x \lhd_i c_i \in G^{nd}\}$. Then, Theorem 3.5 implies that either $v \not\models x \lhd c$ or $v'(x) \leq v(x)$. Now, if $v \not\models x \lhd c$ then $v \not\models x \lhd_i c_i$ for every constraint $x \lhd_i c_i \in G^{nd}$. Therefore, Theorem 3.5 implies $v \sqsubseteq_{\{x \lhd_i c_i\}} v'$ for all $x \lhd_i c_i \in G^{nd}$, which then implies that $v \sqsubseteq_{G^{nd}} v'$ (Proposition 3.3). On the other hand, if $v'(x) \leq v(x)$ then also from Theorem 3.5 it follows that $v \sqsubseteq_{\{x \lhd_i c_i\}} v'$ for every $x \lhd_i c_i \in G^{nd}$. This again proves that $v \sqsubseteq_{G^{nd}} v'$ (thanks to Proposition 3.3). □

Analogous results to Theorem 3.5 and Corollary 3.6 hold for lower-bound non-diagonal constraints as well. The following two results give these two conditions separately. Theorem 3.9 then combines these two results to get the consolidated condition that is necessary and sufficient for the relation $v \sqsubseteq_{\{d \lhd y\}} v'$ to hold.

Lemma 3.7. *Given two valuations v, v' and a non-diagonal constraint $d \lhd y$ such that $v \models d \lhd y$, the relation $v \sqsubseteq_{\{d \lhd y\}} v'$ holds iff $v' \models d \lhd y$.*

Proof. ($\Rightarrow$) This direction follows from Definition 3.1.

($\Leftarrow$) If $v' \models d \lhd y$, then for every $\delta \in \mathbb{R}_{\geq 0}$, $v' + \delta \models d \lhd y$ as well. Therefore, in this case, the relation $v \sqsubseteq_{\{d \lhd y\}} v'$ holds trivially. □

Lemma 3.8. *Given two valuations v, v' and a non-diagonal constraint $d \lhd y$ such that $v \not\models d \lhd y$, the relation $v \sqsubseteq_{\{d \lhd y\}} v'$ holds iff $v(y) \leq v'(y)$.*

Proof. ($\Leftarrow$) From the assumption that $v(y) \leq v'(y)$, it follows that for every $\delta \in \mathbb{R}_{\geq 0}$, $v(y) + \delta \leq v'(y) + \delta$. Then, whenever $d \lhd v(y) + \delta$, it also holds that $d \lhd v'(y) + \delta$. That is, if $v + \delta \models d \lhd y$ then $v' + \delta \models d \lhd y$ as well. This implies $v \sqsubseteq_{\{d \lhd y\}} v'$.

($\Rightarrow$) Assume the relation $v \sqsubseteq_{\{d \lhd y\}} v'$ holds. Now, if $v(y) > v'(y)$ then by the density of real numbers, there exists $\delta \in \mathbb{R}_{\geq 0}$ such that $v(y) + \delta > d$ but $v'(y) + \delta \leq d$. This contradicts the fact that $v \sqsubseteq_{\{d \lhd y\}} v'$. Therefore, $v(y) \leq v'(y)$ must hold. □

The following theorem combines the previous two results to get the condition that is necessary as well as sufficient for the relation $v \sqsubseteq_{\{d \lhd y\}} v'$ to hold.

Theorem 3.9. *Given two valuations v, v' and a non-diagonal constraint $d \lhd y$ where $d \in \mathbb{N}$, the relation $v \sqsubseteq_{\{d \lhd y\}} v'$ holds iff either $v' \models d \lhd y$ or $v(y) \leq v'(y)$.*

Proof. ($\Leftarrow$) If $v' \models d \lhd y$ then the result follows from Lemma 3.7 and on the other hand, if $v(y) \leq v'(y)$ then the result follows from Lemma 3.8.

($\Rightarrow$) For the valuation v, either $v \models d \lhd y$ or $v \not\models d \lhd y$. In the first case, the result follows from Lemma 3.7 and in the second case, the result follows from Lemma 3.8. □

Similar to Corollary 3.6, the following result proves that it is also sufficient to keep only one lower bound per clock in G^{nd}, for the sake of the relation $\sqsubseteq_{G^{nd}}$.

Corollary 3.10. *Let $G^{nd} = \{d_1 \lhd_1 y, d_2 \lhd_2 y, \ldots, d_k \lhd_k y\}$ be a set of non-diagonal lower bound constraints and v, v' be two valuations. Then, $v \sqsubseteq_{G^{nd}} v'$ iff $v \sqsubseteq_{\{d \lhd y\}} v'$, where $d = \max\{d_i \mid d_i \lhd_i y \in G^{nd}\}$ and $\lhd = <$ if $d < y \in G^{nd}$, otherwise $\lhd = \leq$.*

Proof. ($\Rightarrow$) Since the constraint $d \triangleleft y \in G^{nd}$, Proposition 3.3 implies $v \sqsubseteq_{\{d \triangleleft y\}} v'$.

($\Leftarrow$) Assume $v \sqsubseteq_{\{d \triangleleft y\}} v'$. Then, Theorem 3.9 implies that either $v' \models d \triangleleft y$ or $v(y) \leq v'(y)$. If $v' \models d \triangleleft y$ then from the choice of d and $\triangleleft$ it follows that $v' \models d_i \triangleleft_i y$ for every constraint $d_i \triangleleft_i y \in G^{nd}$. Therefore, Theorem 3.9 implies $v \sqsubseteq_{\{d_i \triangleleft_i y\}} v'$ for all $d_i \triangleleft_i y \in G^{nd}$, which then implies that $v \sqsubseteq_{G^{nd}} v'$ (due to Proposition 3.3). On the other hand, if $v(y) \leq v'(y)$ then also from Theorem 3.9 it follows that $v \sqsubseteq_{\{d_i \triangleleft_i y\}} v'$ for every $d_i \triangleleft_i y \in G^{nd}$. Then, Proposition 3.3 again proves that $v \sqsubseteq_{G^{nd}} v'$. □

The LU-preorder $\sqsubseteq_{LU}$ defined by Behrmann et al. in [BBLP06] is recalled below. This relation is defined with respect to two bounds functions, L and U. Each of these functions associates non-negative integers (or $-\infty$) to every clock.

Definition 3.11 (LU-preorder [BBLP06])**.** *Given two valuations v, v' and two bounds functions $L : X \to \mathbb{N} \cup \{-\infty\}$ and $U : X \to \mathbb{N} \cup \{-\infty\}$, the relation $v \sqsubseteq_{LU} v'$ is said to hold if for every clock $x \in X$:*

- *if $v(x) > v'(x)$ then $v'(x) > L(x)$,*
- *if $v'(x) > v(x)$ then $v(x) > U(x)$.*

Note that, the relation $\sqsubseteq_{LU}$ can be defined for all possible bounds functions L, U. In practice, these L, U bounds are defined based on the set of all (or "relevant") guards present in the input automaton. However, these bounds can in fact be defined given any finite set of non-diagonal constraints, in particular, given a set G^{nd}. Two bounds functions $L_{G^{nd}}$ and $U_{G^{nd}}$ are defined below, based on G^{nd}.

Definition 3.12 (LU bounds corresponding to G^{nd})**.** *Given a finite set of non-diagonal constraints G^{nd}, the two functions $L_{G^{nd}} : X \to \mathbb{N} \cup \{-\infty\}$ and $U_{G^{nd}} : X \to \mathbb{N} \cup \{-\infty\}$ are defined as follows, with the convention $\max(\emptyset) = -\infty$:*

- *$L_{G^{nd}}(y) = \max\{d \mid d \triangleleft y \in G^{nd}\}$, for every clock $y \in X$,*
- *$U_{G^{nd}}(x) = \max\{c \mid x \triangleleft c \in G^{nd}\}$, for every clock $x \in X$.*

Finally, the following theorem shows that if two valuations are related with respect to the relation $\sqsubseteq_{LU(G^{nd})}$ then they will also be related with respect to $\sqsubseteq_{G^{nd}}$, when $LU(G^{nd})$ denotes the pair of functions $L_{G^{nd}}$ and $U_{G^{nd}}$ of Definition 3.12.

Theorem 3.13. *Given a finite set of non-diagonal constraints G^{nd}, let $L_{G^{nd}}, U_{G^{nd}}$ be the bounds functions defined from G^{nd} according to Definition 3.12. Then, given a pair of valuations v, v', the relation $v \sqsubseteq_{LU(G^{nd})} v'$ implies the relation $v \sqsubseteq_{G^{nd}} v'$.*

Proof. Let φ be an arbitrary constraint belonging to G^{nd}. To prove the theorem it is sufficient to show that $v \sqsubseteq_{\{\varphi\}} v'$ (due to Proposition 3.3). There are two possibilities for the atomic constraint φ, it can either be of the form $x \triangleleft c$ or of the form $d \triangleleft y$, where c, d are non-negative integers, x, y are clocks and $\triangleleft \in \{<, \leq\}$.

Case 1. Let $\varphi = x \triangleleft c$ for some clock x and some constant $c \in \mathbb{N}$. Then, from Definition 3.12 it follows that $c \leq U_{G^{nd}}(x)$. There are, again, two possibilities: either (i) $v'(x) \leq v(x)$ - in this case, Theorem 3.5 implies that $v \sqsubseteq_{\{\varphi\}} v'$, or (ii) $v'(x) > v(x)$ - in this case Definition 3.11 implies that $U_{G^{nd}}(x) < v(x)$. Since

$c \leq U_{G^{nd}}(x)$, this means that $v \not\models x \triangleleft c$, that is, $v \not\models \varphi$. Then, Theorem 3.5 implies that $v \sqsubseteq_{\{\varphi\}} v'$.

Case 2. Let $\varphi = d \triangleleft y$ for some clock y and some constant $d \in \mathbb{N}$. Then, from Definition 3.12: $d \leq L_{G^{nd}}(y)$. There are two possibilities: either (i) $v(y) \leq v'(y)$ - in this case Theorem 3.9 implies that $v \sqsubseteq_{\{\varphi\}} v'$, or (ii) $v(y) > v'(y)$ - in this case since $v \sqsubseteq_{LU(G^{nd})} v'$, from Definition 3.11 it follows that $L(y) < v'(y)$. Since $d \leq L_{G^{nd}}(y)$, it then follows that $v' \models d \triangleleft y$. Therefore, Theorem 3.9 implies that $v \sqsubseteq_{\{\varphi\}} v'$.

Since φ was an arbitrarily chosen constraint from G^{nd}, it follows that for every constraint $\varphi \in G^{nd}$, $v \sqsubseteq_{\{\varphi\}} v'$. Therefore, Proposition 3.3 implies $v \sqsubseteq_{G^{nd}} v'$. □

However, the reverse direction of Theorem 3.13 does not hold.

Example 3.2.1. This example stems from the observation that, the statement "either $v \not\models x \triangleleft c$ or $v'(x) \leq v(x)$" in Theorem 3.5 can be rephrased as "if $v'(x) > v(x)$ then $v \not\models x \triangleleft c$". Consider the set $G^{nd} = \{x < 1\}$ and the pair of valuations v, v' such that $v(x) = 1$ and $v'(x) = 1.1$. The $L_{G^{nd}}, U_{G^{nd}}$ functions defined over G^{nd}, according to Definition 3.12, are such that $L_{G^{nd}}(x) = -\infty$ and $U_{G^{nd}}(x) = 1$. Now, from the construction of the valuations v, v', it can be seen that $v'(x) > v(x)$, but $v(x) = U_{G^{nd}}(x)$. Therefore, from the definition of $\sqsubseteq_{LU}$ in Definition 3.11, it follows that $v \not\sqsubseteq_{LU(G^{nd})} v'$. However, since $v(x) = 1 \not< 1$, it follows that $v \not\models x < 1$. Therefore, from Theorem 3.5 (with the statement being rephrased as mentioned at the beginning of this example) it follows that $v \sqsubseteq_G v'$.

Theorem 3.13 and Example 3.2.1 together show that the relation $\sqsubseteq_{G^{nd}}$ can relate more valuations than the relation $\sqsubseteq_{LU(G^{nd})}$, where the set G^{nd} contains only non-diagonal constraints and $LU(G^{nd})$ denotes the bounds $L_{G^{nd}}, U_{G^{nd}}$, defined based on G^{nd}, according to Definition 3.12. This is one motivation for studying the relation $\sqsubseteq_G$ defined in the previous section. This improvement is due to the fact that the $L_{G^{nd}}, U_{G^{nd}}$ bounds fail to distinguish between the inequalities $<$ and $\leq$.

Theorem 3.13 extends naturally to zones as well. It follows directly from Definition 3.4 that if two zones satisfy $Z \sqsubseteq_{LU(G^{nd})} Z'$ then $Z \sqsubseteq_{G^{nd}} Z'$ as well.

Corollary 3.14. *Given a finite set of non-diagonal constraints G^{nd}, let $L_{G^{nd}}, U_{G^{nd}}$ be the bounds functions defined from G^{nd} according to Definition 3.12. Then, given a pair of zones Z, Z', if $Z \sqsubseteq_{LU(G^{nd})} Z'$ then $Z \sqsubseteq_{G^{nd}} Z'$ as well.*

Similar to the case with valuations, the reverse direction of the above result need not hold in general. This implies that when the relation $\sqsubseteq_{G^{nd}}$ is used instead of $\sqsubseteq_{LU}$ in a reachability algorithm for diagonal-free timed automata, the computed zone graph is expected to be smaller. However, this improvement of $\sqsubseteq_{G^{nd}}$ over $\sqsubseteq_{LU}$ is not drastic and therefore it may not show large improvements in practice. However, the simulation relation $\sqsubseteq_{LU}$ is defined only for diagonal-free timed automata. Whereas, the relation $\sqsubseteq_G$ can be made a simulation relation for timed automata, even with the presence of diagonal constraints and further for Updatable Timed Automata in general (Section 3.3). One question may arise at this point: one reason for the popularity of the $\sqsubseteq_{LU}$ simulation is the ease of checking this relation, does it become more difficult to check $\sqsubseteq_G$? The answer to this question is delayed till Section 4.2.

3.3 Making $\sqsubseteq_G$ a simulation relation

The aim in this section is to make the relation $\sqsubseteq_G$ a simulation relation, when G can also contain diagonal constraints. Not all choices for the set G results in achieving this aim. This section starts off with a simple construction that is sufficient to ensure $\sqsubseteq_G$ becomes a simulation relation for Timed Automata. This construction is then lifted naturally to incorporate updates. This construction makes $\sqsubseteq_G$ a simulation relation for Updatable Timed Automata as well. However, the parameter becomes impossible to compute for large number of such automata. An improved (and a little more technical) construction is then provided that can be computed for strictly more automata, enabling $\sqsubseteq_G$ to be used for more updatable timed automata.

A simulation relation (Definition 2.16) is, first of all, a relation between two configurations. The relation $\sqsubseteq_G$ has only been defined between two valuations in Definition 3.1, the following definition extends this to a relation between two configurations. Two configurations are related with respect to $\sqsubseteq_G$, only when the states present in both the configurations are the same. Also, in the spirit of the state-dependent clock bounds introduced by Behrmann et al. in [BBFL03], state specific sets of atomic constraints are used below while defining this relation.

Definition 3.15 (the relation $\sqsubseteq_{\mathcal{G}}$ over configurations of UTA)**.** *Given two configurations (q, v) and (q', v') of an updatable timed automaton and a map $\mathcal{G} : q \mapsto \mathcal{G}(q)$ associating sets of atomic constraints to states of the automaton, the relation $(q, v) \sqsubseteq_{\mathcal{G}} (q', v')$ holds if (i) $q = q'$ and (ii) $v \sqsubseteq_{\mathcal{G}(q)} v'$.*

The goal now is to compute these sets $\mathcal{G}(q)$ for every state q of the underlying automaton, so that, the resulting relation $\sqsubseteq_{\mathcal{G}}$ becomes a simulation relation. Before proceeding towards this goal, recall that the reachability algorithm (Algorithm 1) does not deal with configurations and instead deals with nodes of the form (q, Z) where q is a state of the automaton and Z is a zone. Similar to the above Definition 3.15, the relation $\sqsubseteq_G$ defined between zones in Definition 3.4 can be extended to a relation between two nodes (having the same state) in the following manner.

Definition 3.16 (the relation $\sqsubseteq_{\mathcal{G}}$ over nodes of zone graph)**.** *Given two nodes (q, Z) and (q', Z') of the zone graph of an updatable timed automaton and a map $\mathcal{G} : q \mapsto \mathcal{G}(q)$, $(q, Z) \sqsubseteq_{\mathcal{G}} (q, Z')$ if (i) $q = q'$ and (ii) $Z \sqsubseteq_{\mathcal{G}(q)} Z'$.*

The above definition can also be rewritten in terms of Definition 3.15: the relation $(q, Z) \sqsubseteq_{\mathcal{G}} (q, Z')$ holds if for every (q, v) with $v \in Z$, there exists (q, v') where $v' \in Z'$ such that $(q, v) \sqsubseteq_{\mathcal{G}} (q, v')$. Thus, for every state q, the set $\mathcal{G}(q)$ used in the definition for $(q, v) \sqsubseteq_{\mathcal{G}} (q, v')$ is the same set used to define $(q, Z) \sqsubseteq_{\mathcal{G}} (q, Z')$. The following Section 3.3.1 describes a simple construction of such a map $\mathcal{G} : q \mapsto \mathcal{G}(q)$, that makes $\sqsubseteq_{\mathcal{G}}$ a simulation relation for Timed Automata.

The rest of this chapter is devoted towards constructing the appropriate parameter in order to make the relation $\sqsubseteq_{\mathcal{G}}$, defined over configurations, a simulation relation (Definition 2.16). The rest of this chapter, therefore, considers only the relations defined between valuations (Definition 3.1) and configurations (Definition 3.15). Chapter 4 contains further discussions regarding the relations over zones (Definition 3.4) and nodes (Definition 3.16).

3.3.1 Constructing an appropriate parameter: initial try

For a start, this section considers only (classical) Timed Automata. Therefore, this section considers only resets, that is, updates of the form $x := 0$, where x is a clock. To make this restriction more evident, in this section from now on the notation for updates is reverted back to R (to denote resets) from *up*. More general updates will be considered in the next section. This section proposes a construction for the map $\mathcal{G}$, that is sufficient to ensure $\sqsubseteq_{\mathcal{G}}$ is a simulation relation for Timed Automata. The results presented in this section are part of [GMS19].

The conditions required to be satisfied by a relation in order to be a simulation relation (Definition 2.16) are rewritten below for the case of Timed Automata. In order to be a simulation relation, $\sqsubseteq_{\mathcal{G}}$ (Definition 3.15) needs to satisfy the following conditions whenever two configurations (q, v) and (q, v') satisfy $(q, v) \sqsubseteq_{\mathcal{G}} (q, v')$:

1. for every $\delta \geq 0$, $(q, v + \delta) \sqsubseteq_{\mathcal{G}} (q, v' + \delta)$,
2. for every transition $t = (q, g, R, q')$ of $\mathcal{A}$, if $v \models g$ then –
   (a) $v' \models g$, and
   (b) $(q', [R]v) \sqsubseteq_{\mathcal{G}} (q', [R]v')$.

Among the three conditions above, Condition 1 follows directly from Definition 3.1 and therefore does not require any special construction for $\sqsubseteq_{\mathcal{G}}$. For every choice of $\mathcal{G}$, the relation $\sqsubseteq_{\mathcal{G}}$ satisfies this condition. This is proved below.

Lemma 3.17. *If* $(q, v) \sqsubseteq_{\mathcal{G}} (q, v')$ *then for every* $\delta \geq 0$, $(q, v + \delta) \sqsubseteq_{\mathcal{G}} (q, v' + \delta)$.

Proof. Assume $(q, v) \sqsubseteq_{\mathcal{G}} (q, v')$, that is, $v \sqsubseteq_{\mathcal{G}(q)} v'$. It needs to be shown that $(v+\delta) \sqsubseteq_{\mathcal{G}(q)} (v'+\delta)$. Consider an atomic constraint $\varphi \in \mathcal{G}(q)$. Let $\delta' \in \mathbb{R}_{\geq 0}$ be such that $(v + \delta) + \delta' \models \varphi$. This can be rewritten as $v + (\delta + \delta') \models \varphi$. Since $v \sqsubseteq_{\mathcal{G}(q)} v'$, it follows that $v' + (\delta + \delta') \models \varphi$ as well. Again, this can be rewritten as $(v' + \delta) + \delta' \models \varphi$. Therefore, the following implication holds $(v + \delta) + \delta' \models \varphi \implies (v' + \delta) + \delta' \models \varphi$. This completes the proof of the lemma. □

Satisfaction of the remaining two conditions, Condition 2a and Condition 2b, indeed require specific constructions of $\mathcal{G}$. One such construction is discussed in this section. First, the following question is considered: what atomic constraints does the set $\mathcal{G}(q)$ need to contain so that whenever $(q, v) \sqsubseteq_{\mathcal{G}} (q, v')$, for every transition (q, g, R, q'), if v satisfies the guard g of the transition then v' also satisfies g?

Ensuring condition 2a. Recall from Definition 3.1 that if $v \sqsubseteq_{\mathcal{G}(q)} v'$, then for every constraint φ belonging to the set $\mathcal{G}(q)$, $v \models \varphi$ implies $v' \models \varphi$. Therefore, if $\mathcal{G}(q)$ contains the guard g itself, then the required condition will be satisfied whenever $(q, v) \sqsubseteq_{\mathcal{G}} (q, v')$. However, the sets $\mathcal{G}(q)$ contain only atomic constraints and g can also be a conjunction of atomic constraints. Thus, g cannot directly be added to $\mathcal{G}(q)$. The following lemma proves that if $\mathcal{G}(q)$ contains each of the atomic constraints present in g, the relation $\sqsubseteq_{\mathcal{G}}$ satisfies the required condition.

Lemma 3.18. *Let $(q, v) \sqsubseteq_{\mathcal{G}} (q, v')$. For every transition $t = (q, g, R, q')$ of $\mathcal{A}$, if $\mathcal{G}(q)$ contains all atomic constraints present in g, then $v \models g$ implies $v' \models g$.*

Proof. Since $(q, v) \sqsubseteq_{\mathcal{G}} (q, v')$, from Definition 3.15 it follows that $v \sqsubseteq_{\mathcal{G}(q)} v'$. Assume that all atomic constraints present in g are included in $\mathcal{G}(q)$ and $v \models g$. Let $g = \varphi_1 \wedge \varphi_2 \wedge \cdots \wedge \varphi_n$, where each φ_i is an atomic constraint. Then, $v \models g \implies v \models \varphi_i$ for all $i \in \{1, 2, \ldots, n\}$. Now, since $v \sqsubseteq_{\mathcal{G}(q)} v'$ and for every i, $\varphi_i \in \mathcal{G}(q)$, Definition 3.1 implies that $v' \models \varphi_i$. Therefore, $v' \models \varphi_1 \wedge \varphi_2 \wedge \cdots \wedge \varphi_n$ as well, that is, $v' \models g$. □

The lemma above provides the first step for constructing the map $\mathcal{G}$:

---

for every transition (q, g, R, q') –
$\mathcal{G}(q)$ contains all atomic constraints present in the guard g (3.1)

Ensuring condition 2b. This condition states that, for every transition $t = (q, g, R, q')$ and for every pair of configurations (q, v) and (q, v'), if the relation $(q, v) \sqsubseteq_{\mathcal{G}} (q, v')$ holds and t is *enabled* from (q, v), then, after taking the transition, the resulting configurations preserve the relation, that is, $(q', [R]v) \sqsubseteq_{\mathcal{G}} (q', [R]v')$. In other words, this condition requires – if $v \sqsubseteq_{\mathcal{G}(q)} v'$ and $v \models g$, then the relation $[R]v \sqsubseteq_{\mathcal{G}(q')} [R]v'$ must also hold. Since this condition involves two sets $\mathcal{G}(q)$ and $\mathcal{G}(q')$, a dependence between these two sets is required in order to satisfy this condition. This is achieved by *propagating* each constraint present in $\mathcal{G}(q')$ to the set $\mathcal{G}(q)$, along the selected transition $t = (q, g, R, q')$. The propagated atomic constraints act as "pre-conditions" that are sufficient to ensure $v \sqsubseteq_{\mathcal{G}(q)} v'$ and $v \models g$ imply $[R]v \sqsubseteq_{\mathcal{G}(q')} [R]v'$. Given a constraint from the set $\mathcal{G}(q')$ and the reset R (present in the selected transition (q, g, R, q')), the atomic constraint that is propagated to the set $\mathcal{G}(q)$, is defined below.

Definition 3.19 (pre under reset)**.** *Given an atomic constraint φ and a set of clocks R, the* pre *of φ under the reset of clocks in R, is the constraint $[R]^{-1}(\varphi)$, where $[R]^{-1}(\varphi) = \varphi[0/x, \forall x \in R]$.*

For example, if $R = \{x, y\}$ then (i) $[R]^{-1}(x \leq 3) = 0 \leq 3 = \top$, (ii) $[R]^{-1}(y > 2) = 0 > 2 = \bot$, (iii) $[R]^{-1}(z \leq 1) = z \leq 1$, (iv) $[R]^{-1}(x - z \leq -2) = 2 \leq z$.

Upcoming Lemma 3.21 proves that for every transition $t = (q, g, R, q')$, if the set $\mathcal{G}(q)$ contains the atomic constraint $[R]^{-1}(\varphi)$, for every constraint $\varphi \in \mathcal{G}(q')$ then the relation $\sqsubseteq_{\mathcal{G}}$ satisfies the condition 2b as well. But before proving this lemma, it will be useful to prove the following intermediate result first. This result considers the restricted case when $\mathcal{G}(q')$ consists only of a single atomic constraint φ instead of a set of atomic constraints. The proof of this result considers the various possibilities for the atomic constraint φ and the set of resets R.

Lemma 3.20. *Given two valuations v, v', an atomic constraint φ and a set of clocks R, if the relation $v \sqsubseteq_{\{[R]^{-1}(\varphi)\}} v'$ holds then $[R]v \sqsubseteq_{\{\varphi\}} [R]v'$ holds.*

Proof. Let $v \sqsubseteq_{\{[R]^{-1}(\varphi)\}} v'$ and $\delta \in \mathbb{R}_{\geq 0}$ be a constant such that $[R]v + \delta \models \varphi$. To prove $[R]v \sqsubseteq_{\{\varphi\}} [R]v'$, it needs to be shown that $[R]v' + \delta \models \varphi$ as well.

Consider the case where φ is a non-diagonal constraint of the form $x \lhd c$. Assume that the clock $x \notin R$. Then, $([R]v)(x) = v(x)$, $([R]v')(x) = v'(x)$ and $[R]^{-1}(\varphi) = x \lhd c$. The following holds: $[R]v + \delta \models x \lhd c \implies ([R]v)(x) + \delta \lhd c \implies v(x) + \delta \lhd c \implies v + \delta \models [R]^{-1}(\varphi)$. Since $v \sqsubseteq_{\{[R]^{-1}(\varphi)\}} v'$, it then follows that $v' + \delta \models x \lhd c$ as well. Following the above sequence of arguments backwards, it can be deduced that $[R]v' + \delta \models x \lhd c$, that is, $[R]v' + \delta \models \varphi$. On the other hand, assume that $x \in R$. In this case, $([R]v)(x) = 0 = ([R]v')(x)$. Therefore, for every $\delta \in \mathbb{R}_{\geq 0}$, $([R]v)(x) + \delta = ([R]v')(x) + \delta$. This implies, $[R]v + \delta \models x \lhd c \iff [R]v' + \delta \models x \lhd c$. The case when $\varphi = d \lhd y$ can be proved using similar arguments.

Now, assume φ be a diagonal constraint $x - y \lhd c$. If $x \notin R$ and $y \in R$, then $([R]v)(x) = v(x)$, $([R]v')(x) = v'(x)$ and $([R]v)(y) = 0 = ([R]v')(y)$. Also, in this case, $[R]^{-1}(\varphi) = x \lhd c$. The following sequence of implications hold: $[R]v + \delta \models \varphi \implies ([R]v)(x) + \delta - (([R]v)(y) + \delta) \lhd c \implies ([R]v)(x) - 0 \lhd c \implies v(x) \lhd c \implies v \models x \lhd c \implies v \models [R]^{-1}(\varphi)$. Since $v \sqsubseteq_{\{[R]^{-1}(\varphi)\}} v'$, it follows that $v' \models [R]^{-1}(\varphi)$. From this, following the above sequence of reasoning backwards, it can be deduced that $[R]v' + \delta \models \varphi$. On the other hand, assume now that $x \in R, y \notin R$. Then, $([R]v)(x) = 0 = ([R]v')(x)$ and $([R]v)(y) = v(y), ([R]v')(y) = v'(y)$. Also, $[R]^{-1}(\varphi) = -y \lhd c = -c \lhd y$. The following hold: $[R]v + \delta \models \varphi \implies [R]v + \delta \models x - y \lhd c \implies ([R]v)(x) + \delta - (([R]v)(y) + \delta) \lhd c \implies -v(y) \lhd c \implies v \models -c \lhd y \implies v \models [R]^{-1}(\varphi)$. Again, since $v \sqsubseteq_{\{[R]^{-1}(\varphi)\}} v'$, it follows that $v' \models [R]^{-1}(\varphi)$ and therefore it can again be deduced, following the previous reasoning backwards, that $[R]v' + \delta \models \varphi$. The remaining two cases are when both $x \in R, y \in R$ or when $x \notin R, y \notin R$. In both of these cases, $([R]v)(x) = ([R]v')(x)$ and $([R]v)(y) = ([R]v')(y)$. Therefore, $[R]v + \delta \models x - y \lhd c \iff [R]v' + \delta \models x - y \lhd c$. When $\varphi = d \lhd x - y$, the result can be proved using similar reasoning. □

The following lemma generalizes the previous lemma by considering $\mathcal{G}(q')$ to be a (not necessarily singleton) set of atomic constraints. This result proves: for every transition $t = (q, g, R, q')$ and for every atomic constraint φ in $\mathcal{G}(q')$ if the constraint $[R]^{-1}(\varphi)$ is added to the set $\mathcal{G}(q)$ then the relation $\sqsubseteq_{\mathcal{G}}$ satisfies condition 2b.

Lemma 3.21. *Let $(q, v) \sqsubseteq_{\mathcal{G}} (q, v')$ and $t = (q, g, R, q')$ be a transition of $\mathcal{A}$. For every constraint $\varphi \in \mathcal{G}(q')$, if $\mathcal{G}(q)$ contains $[R]^{-1}(\varphi)$, then $(q', [R]v) \sqsubseteq_{\mathcal{G}} (q', [R]v')$.*

Proof. Since $(q, v) \sqsubseteq_{\mathcal{G}} (q, v')$, from Definition 3.15 it follows that $v \sqsubseteq_{\mathcal{G}(q)} v'$. It needs to be shown that $[R]v \sqsubseteq_{\mathcal{G}(q')} [R]v'$. Choose a constraint $\varphi \in \mathcal{G}(q')$. From the assumption of this lemma it follows that $\mathcal{G}(q)$ contains the constraint $[R]^{-1}(\varphi)$. Now, since $v \sqsubseteq_{\mathcal{G}(q)} v'$, from Proposition 3.3 it follows that $v \sqsubseteq_{\{[R]^{-1}(\varphi)\}} v'$ as well. Then, from Lemma 3.20 it follows that $[R]v \sqsubseteq_{\{\varphi\}} [R]v'$. Finally, since this holds for every $\varphi \in \mathcal{G}(q)$, again from Proposition 3.3, it follows that $[R]v \sqsubseteq_{\mathcal{G}(q')} [R]v'$. □

$$\begin{gathered}\textbf{for every transition } t = (q, g, R, q') \text{ and} \\ \textbf{for every constraint } \varphi \in \mathcal{G}(q'): \\ \mathcal{G}(q) \text{ contains the atomic constraint } \mathsf{pre}(\varphi, R) = [R]^{-1}(\varphi)\end{gathered} \tag{3.2}$$

Given a timed automaton $\mathcal{A} = (Q, X, q_0, T, F)$, two steps have now been proposed in (3.1) and (3.2), that are required to be taken while constructing a map $\mathcal{G}$, to make the relation $\sqsubseteq_{\mathcal{G}}$ (Definition 3.15) a simulation relation, when defined with respect to such a map $\mathcal{G}$. Such maps will be called $\mathcal{A}$-maps.

Definition 3.22 ($\mathcal{A}$-map)**.** *An $\mathcal{A}$-map is a tuple $\mathcal{G} := (\mathcal{G}(q))_{q\in Q}$, where each $\mathcal{G}(q)$ is a set of atomic constraints associated with the state q, satisfying the following two conditions: for every state q of $\mathcal{A}$ and for every transition $t = (q, g, R, q')$ of $\mathcal{A}$,*

- *$\mathcal{G}(q)$ contains the atomic constraints present in g,*
- *for every $\varphi \in \mathcal{G}(q')$, $\mathcal{G}(q)$ contains the atomic constraint $[R]^{-1}(\varphi)$.*

The following theorem proves that when the relation $\sqsubseteq_{\mathcal{G}}$ in Definition 3.15, relating two configurations, is defined with respect to an $\mathcal{A}$-map $\mathcal{G}$, it becomes a simulation relation for the timed automaton $\mathcal{A}$. The proof of this result follows directly from Lemma 3.17, 3.18 and 3.21.

Theorem 3.23. *Given a timed automaton $\mathcal{A}$, possibly containing diagonal constraints, the relation $\sqsubseteq_{\mathcal{G}}$ (of Definition 3.15) defined over the space of configurations of $\mathcal{A}$ is a simulation relation, if the tuple $(\mathcal{G}(q))_{q\in Q_{\mathcal{A}}}$ is an $\mathcal{A}$-map.*

Proof. In order to be a simulation relation, the relation $\sqsubseteq_{\mathcal{G}}$ in Definition 3.15 needs to satisfy the three conditions 1, 2a and 2b. When the map $\mathcal{G}$ is an $\mathcal{A}$-map of Definition 3.22, Lemma 3.17 proves that $\sqsubseteq_{\mathcal{G}}$ satisfies the condition 1, Lemma 3.18 implies that $\sqsubseteq_{\mathcal{G}}$ satisfies the condition 2a and finally Lemma 3.21 ensures $\sqsubseteq_{\mathcal{G}}$ satisfies the condition 2b. Therefore, when $\mathcal{G}$ is an $\mathcal{A}$-map, $\sqsubseteq_{\mathcal{G}}$ is a simulation relation. □

Thanks to the theorem above, the problem of finding a simulation relation for timed automata now reduces to computing an $\mathcal{A}$-map given a timed automaton $\mathcal{A}$. An $\mathcal{A}$-map can be computed by computing the least fixpoint of a system of equations, as defined below.

Definition 3.24 (state based guards)**.** *Let $\mathcal{A} = (Q, X, q_0, T, F)$ be a given timed automaton. Associate a set of atomic constraints $\mathcal{G}(q)$ to every state $q \in Q$, where $\{\mathcal{G}(q)\}_{q\in Q}$ is the least solution to the following set of equations and* $\mathsf{atoms\_of}(g)$ *denotes the set of all atomic constraints present in the constraint g:*

$$\mathcal{G}(q) = \bigcup_{(q,g,R,q')\in T} \left\{ \mathsf{atoms\_of}(g) \cup \left(\bigcup_{\varphi\in\mathcal{G}(q')} \mathsf{pre}(\varphi, R) \right) \right\}$$

Algorithm 2 computes the least (with respect to pointwise set inclusion order) fixpoint of the set of equations present in Definition 3.24. Given a timed automaton $\mathcal{A}$, this algorithm computes the sets $\mathcal{G}(q)$ corresponding to every state q of the input automaton $\mathcal{A}$ such that these sets satisfy the two conditions derived from Lemma 3.18 and Lemma 3.21. Therefore, the following Algorithm 2 computes $\mathcal{A}$-map. This is stated in Theorem 3.25.

Input: $\mathcal{A} = (Q, X, q_0, T, F)$
Output: $\mathcal{G}(q)$ for every state $q \in Q$

foreach *state* $q \in Q$ **do**
  $\mathcal{G}(q) \leftarrow \emptyset$
end
foreach *transition* $t = (q, g, R, q') \in T$ **do**
  $\mathcal{G}(q) \leftarrow \mathcal{G}(q) \cup \mathsf{atoms\_of}(g)$; `// atoms_of(g) is the set of all`
  `// atomic constraints present in` g
end
while *a fixed point of* $\{\mathcal{G}(q)\}_{q \in Q}$ *is not reached* **do**
  foreach *transition* $t = (q, g, R, q') \in T$ **do**
    foreach *constraint* φ *in* $\mathcal{G}(q')$ **do**
      $\mathcal{G}(q) \leftarrow \mathcal{G}(q) \cup \{[R]^{-1}(\varphi)\}$;
    end
  end
end

Algorithm 2: Computing state based guards $\mathcal{G}(q)$

Theorem 3.25. *Given a timed automaton* $\mathcal{A} = (Q, X, q_0, T, F)$*, every set* $\{\mathcal{G}(q)\}_{q \in Q}$ *that either satisfies the set of equations of Definition 3.24 or that is generated by Algorithm 2 is an* $\mathcal{A}$*-map.*

Example 3.3.1. The $\mathcal{A}$-map computation according to Algorithm 2 for the automaton given in Figure 3.1, is given in Figure 3.2:

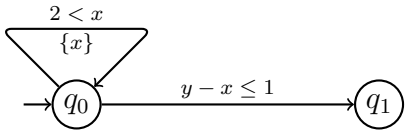


Figure 3.1: Example $\mathcal{A}$-map computation

$$\begin{array}{lccccc} \mathcal{G}(q_0) = & \{\} & \longrightarrow & \{2 < x, y - x \leq 1\} & \longrightarrow & \{2 < x, y - x \leq 1, y \leq 1\} \\ \mathcal{G}(q_1) = & \{\} & & \{\} & & \{\} \end{array}$$

Figure 3.2: $\mathcal{A}$-map computation for the automaton in Figure 3.1

Note that, given a timed automaton $\mathcal{A}$, none of the steps in Algorithm 2 generates a constraint with a *new* constant. By a "new" constant it is meant that, a constant that is not present in any of the guards of $\mathcal{A}$. This ensures that Algorithm 2 terminates on all input (timed automaton) $\mathcal{A}$.

Theorem 3.26. *Given a timed automaton* $\mathcal{A}$*, Algorithm 2 computing the* $\mathcal{A}$*-map corresponding to* $\mathcal{A}$*, is guaranteed to terminate.*

Before concluding this section, as an aside, note that, the reverse direction of Lemma 3.20 does not hold in general. It can be proved that it holds whenever the constraints φ and $[R]^{-1}(\varphi)$ are either both non-diagonal constraints or both diagonal constraints. It however may not hold if φ is a diagonal constraint and $[R]^{-1}(\varphi)$ is a non-diagonal constraint.

An example where the reverse direction of Lemma 3.20 does not hold is as follows: consider $\varphi = x - y \leq 5$, let R be the set $\{y\}$ and v, v' be two valuations such that $v(x) = 4, v(y) = 2$ and $v'(x) = 4.5, v'(y) = 3$. Then, $[R]^{-1}(\varphi) = x \leq 5$. If $\delta = 1$ then $v(x) + \delta = 4 + 1 = 5 \leq 5$, that is $v + \delta \models [R]^{-1}(\varphi)$ but $v'(x) + \delta = 4.5 + 1 = 5.5 \not\leq 5$, that is $v' + \delta \not\models [R]^{-1}(\varphi)$. Therefore, $v \not\sqsubseteq_{\{[R]^{-1}(\varphi)\}} v'$. However, $([R]v)(x) = 4, ([R]v)(y) = 0$, thus $[R]v \models \varphi$ and also $([R]v')(x) = 4.5, ([R]v')(y) = 0$, thus $[R]v' \models \varphi$. Since φ is a diagonal constraint, for every constant $\delta \in \mathbb{R}_{\geq 0}$, both $[R]v + \delta \models \varphi$ and $[R]v' + \delta \models \varphi$ hold. Therefore, $[R]v \sqsubseteq_{\{\varphi\}} [R]v'$, proving that $[R]v \sqsubseteq_{\{\varphi\}} [R]v' \not\Rightarrow v \sqsubseteq_{\{[R]^{-1}(\varphi)\}} v'$.

This shows that, in order to ensure the relation $[R]v \sqsubseteq_{\{\varphi\}} [R]v'$, it is sufficient *but not necessary* for the relation $v \sqsubseteq_{\{[R]^{-1}(\varphi)\}} v'$ to hold. Therefore, the construction (3.2), derived from Lemma 3.21, can possibly be done away with in certain situations. It will be shown in the next section that such an optimization is indeed possible – and, in fact, somewhat necessary – in the case of Updatable Timed Automata. This optimization stems from another observation. Note that, although condition 2b only talks about the configurations (q, v) where the valuation v satisfies the guard of the transition, Lemma 3.20 and Lemma 3.21 are oblivious to this information. Considering this additional information brings some technical challenges. The next and the subsequent section discuss more about these.

3.3.2 Constructing an appropriate parameter: better try

The previous section considered only (classical) Timed Automata and proved that the relation $\sqsubseteq_{\mathcal{G}}$ of Definition 3.15 becomes a simulation relation for a timed automaton $\mathcal{A}$ when the map $\mathcal{G}$ is an $\mathcal{A}$-map (of Definition 3.22). This section considers updates and the goal is to come up with a map $\mathcal{G} : q \mapsto \mathcal{G}(q)$ that will make the relation $\sqsubseteq_{\mathcal{G}}$ (Definition 3.15) a simulation relation for the class of Updatable Timed Automata. Two constructions will be discussed in this section. The first is the natural extension of the construction presented in the previous section, with resets replaced with updates. This construction turns out to be sufficient to ensure $\sqsubseteq_{\mathcal{G}}$ becomes a simulation relation, however, this construction is not optimal. For a large number of updatable timed automata this construction can result in making one of its $\mathcal{G}(q)$ infinite. Therefore, a better construction is required. This section concludes with this construction, which results in maps $\mathcal{G}$ with smaller $\mathcal{G}(q)$ sets. Proposition 3.2 implies that these (new) maps $\mathcal{G}$ will result in coarser relations. The first construction presented in this section is part of [GMS19] and the improved construction presented later in this section is part of [GMS20].

A simulation relation needs to satisfy a slightly modified set of conditions in the presence of *updates* than in the presence of only resets (as presented on Page 38). The former needs to satisfy the conditions 1 and 2a as the latter along with a modified version of condition 2b: for every transition $t = (q, g, up, q')$, if $(q, v) \sqsubseteq (q, v')$

and t is enabled from (q, v), that is, $v \models g$ and $up(v)(x) \geq 0$ for every clock x, then $(q', up(v)) \sqsubseteq (q', up(v'))$ needs to hold instead of $(q', [R]v) \sqsubseteq (q', [R]v')$ as in the case of resets. To recall a notation: if for every clock x, $up(v)(x) \geq 0$ then $up(v) \geq 0$. Now, to put the above three conditions in one place, in order to be a simulation relation, $\sqsubseteq_{\mathcal{G}}$ needs to satisfy the following: if (q, v) and (q, v') are two configurations of an updatable timed automaton such that $(q, v) \sqsubseteq_{\mathcal{G}} (q, v')$, then -

1. for every $\delta \geq 0$, $(q, v + \delta) \sqsubseteq_{\mathcal{G}} (q, v' + \delta)$,
2. for every transition $t = (q, g, up, q')$ of $\mathcal{A}$, if $v \models g$ and $up(v) \geq 0$ then –
   (a) $v' \models g$,
   (b) $up(v') \geq 0$, and
   (c) $(q', up(v)) \sqsubseteq_{\mathcal{G}} (q', up(v'))$.

Condition 1 follows directly from Definition 3.1. Since condition 2a is unchanged, the same construction (3.1) mentioned on Page 39 for the sets $\mathcal{G}(q)$ is sufficient to ensure this condition is satisfied by $\sqsubseteq_{\mathcal{G}}$ (Lemma 3.18).

Satisfying Condition 2b. Unlike resets, not all updates result in a valid valuation. For example, consider v to be a valuation with $v(x) = 2$ and let $up_x = x - 3$. Then, $(up(v))(x) = -1 < 0$ and therefore $up(v)$ is not a valid valuation, since $up(v) \not\geq 0$. Since this phenomenon was not present in the case of resets, the construction for the sets $\mathcal{G}(q)$ need the following additional step.

$$\begin{gathered}\textbf{for every transition } t = (q, g, R, q') \text{ and} \\ \textbf{for every clock } x \in X\text{:} \\ \mathcal{G}(q) \text{ contains the atomic constraint } up^{-1}(0 \leq x)\end{gathered} \tag{3.3}$$

Lemma 3.27. *Given two valuations v, v' and a transition $t = (q, g, up, q')$, if the set $\mathcal{G}(q)$ contains the constraint $up^{-1}(0 \leq x)$ for every clock x, then the relation $v \sqsubseteq_{\mathcal{G}(q)} v'$ and $up(v) \geq 0$ imply that $up(v') \geq 0$ as well.*

The proof of this lemma follows from Definition 3.1.

Satisfying Condition 2c. A slightly modified condition calls for a slightly modified construction. In the case of Timed Automata, the following construction was required to ensure $\sqsubseteq_{\mathcal{G}}$ satisfies condition 2b (as given on Page 40): for every transition $t = (q, g, R, q')$ and for every constraint $\varphi \in \mathcal{G}(q')$, the set $\mathcal{G}(q)$ contains the "pre-condition" $[R]^{-1}(\varphi)$. This construction can be naturally extended to the case of updates in the following way: for every transition $t = (q, g, up, q')$ and for every constraint $\varphi \in \mathcal{G}(q')$, the set $\mathcal{G}(q)$ contains the constraint $up^{-1}(\varphi)$. Turns out, this construction is sufficient to ensure the relation $\sqsubseteq_{\mathcal{G}}$ satisfies the condition 2c. Before proving this, the constraint $up^{-1}(\varphi)$ is formally defined below.

To recall, an update up is a function $x \mapsto up_x$, where x is a clock and up_x is either a constant $c \in \mathbb{N}$ or some clock y (possibly x) or the expression $y + d$ where y is a non-zero clock (can be x) and d is a positive or negative integer. Note that, all these three kinds of updates can be expressed in general by the expression $z + d$, where z is a clock (allowed to be the zero clock) and d is a positive or negative integer. Therefore, the proofs that are present in this section, will assume, without any loss of generality, that an update to a clock x, that is up_x, is always of the form $z + d$, where z is a clock (possibly the zero clock) and $d \in \mathbb{Z}$.

Definition 3.28 (pre under update)**.** *Given an atomic constraint φ and an update up, the* pre *of φ under the update up, is the constraint $up^{-1}(\varphi)$, where $up^{-1}(\varphi) = \varphi[up_x/x, \forall x \in X]$.*

Following are a few examples of the constraint $up^{-1}(\varphi)$ mentioned in the above definition: (i) let $\varphi = x < 2$ and $up_x = y$ then $up^{-1}(\varphi) = y < 2$, (ii) let $\varphi = x - y < 2$ and $up_x = x$ (that is, no update to x) and $up_y = z + 2$ then $up^{-1}(\varphi) = x - (z + 2) < 2 = x - z < 4$, (iii) let $\varphi = x - y < 2$ and $up_x = z_1 + 5$ and $up_y = z_2 + 2$, then $up^{-1}(\varphi) = (z_1 + 5) - (z_2 + 2) < 2 = z_1 - z_2 < 2 - 3 = z_1 - z_2 < -1 = 1 < z_2 - z_1$, (iv) let $\varphi = x - y \leq 2$ and $up_x = 1$ and $up_y = z$, then $up^{-1}(\varphi) = 1 - z \leq 2 = -z \leq 1 = -1 \leq z$, which is actually the *trivial* atomic constraint $\top$ since this constraint is always *true*.

The following is a generalization of Lemma 3.20. This states that, for an atomic constraint φ, if two valuations are related with respect to $\sqsubseteq_{\{up^{-1}(\varphi)\}}$ then the updated valuations are related with respect to $\sqsubseteq_{\{\varphi\}}$. The proof involves considering several possibilities for the constraint φ and the update up.

Lemma 3.29. *Given two valuations v, v', an atomic constraint φ and an update up such that $up(v) \geq 0$ and $up(v') \geq 0$, if $v \sqsubseteq_{\{up^{-1}(\varphi)\}} v'$ then $up(v) \sqsubseteq_{\{\varphi\}} up(v')$.*

Proof. Assume $v \sqsubseteq_{\{up^{-1}(\varphi)\}} v'$ and $\delta \in \mathbb{R}_{\geq 0}$ is a constant such that $up(v) + \delta \models \varphi$. The aim is to show that $up(v') + \delta \models \varphi$.

First, consider φ to be a diagonal constraint. Then, $up(v) + \delta \models \varphi \iff up(v) \models \varphi$ holds. Now, let φ be of the form $x - y \lhd c$. Then, $up(v) \models \varphi \implies up(v)(x) - up(v)(y) \lhd c$. If $up_x = z_1 + d_1$ and $up_y = z_2 + d_2$ then $up(v)(x) - up(v)(y) = v(z_1) + d_1 - v(z_2) - d_2$ and $up^{-1}(\varphi) = (z_1 + d_1) - (z_2 + d_2) \lhd c$. Hence, $up(v)(x) - up(v)(y) \lhd c \implies v \models (z_1 + d_1) - (z_2 + d_2) \lhd c$, that is, $v \models up^{-1}(\varphi)$. Since $v \sqsubseteq_{\{up^{-1}(\varphi)\}} v'$, it follows that $v' \models (z_1 + d_1) - (z_2 + d_2) \lhd c$ as well. From this, it can be derived that $up(v') \models \varphi$. Again, since φ is a diagonal, $up(v') \models \varphi \iff up(v') + \delta \models \varphi$.

Now, consider φ to be a non-diagonal constraint of the form $x \lhd c$. Let $up_x = d$ for some $d \in \mathbb{N}$. Then, $(up(v))(x) = d = (up(v'))(x)$. Therefore, $up(v) + \delta \models x \lhd c \iff up(v') + \delta \models x \lhd c$. Otherwise, let $up_x = z + d$ where z is a clock and d is a (positive or negative) integer. Then $up^{-1}(\varphi) = z \lhd c - d$ and $(up(v))(x) = v(z) + d$ and $(up(v'))(x) = v'(z) + d$. If $c - d < 0$ then $up^{-1}(\varphi) = z \lhd c - d = \top$ and hence $v + \delta \models z \lhd c - d$ holds trivially. Otherwise, the following hold: $up(v) + \delta \models \varphi \implies up(v) + \delta \models x \lhd c \implies (up(v))(x) + \delta \lhd c \implies v(z) + d + \delta \lhd c \implies v + \delta \models z \lhd c - d$. Since $v \sqsubseteq_{\{z \lhd c - d\}} v'$, it then follows

that $v' + \delta \models z \lhd c - d \implies (v'(z) + d) + \delta \lhd c \implies (up(v'))(x) + \delta \lhd c \implies up(v') + \delta \models x \lhd c \implies up(v') + \delta \models \varphi$. Similar arguments hold when φ is of the form $c \lhd y$ as well. Therefore for every atomic constraint φ, the relation $v \sqsubseteq_{\{up^{-1}(\varphi)\}} v'$ implies $up(v) \sqsubseteq_{\{\varphi\}} up(v')$. □

The following lemma extends the previous one by letting $\mathcal{G}(q')$ be a set containing possibly more than one atomic constraint. This next lemma states that, given a transition (q, g, up, q'), for every constraint $\varphi \in \mathcal{G}(q')$, adding the constraint $up^{-1}(\varphi)$ to the set $\mathcal{G}(q)$, is sufficient to ensure $(q, v) \sqsubseteq_{\mathcal{G}} (q, v')$ implies $(q', up(v)) \sqsubseteq_{\mathcal{G}} (q', up(v'))$. The proof follows from Definition 3.15 and Lemma 3.29.

Lemma 3.30. *Let $(q, v) \sqsubseteq_{\mathcal{G}} (q, v')$ and $t = (q, g, up, q')$ be a transition of the updatable timed automaton $\mathcal{A}$. For every constraint $\varphi \in \mathcal{G}(q')$, if $\mathcal{G}(q)$ contains the constraint $up^{-1}(\varphi)$, then $(q', up(v)) \sqsubseteq_{\mathcal{G}} (q', up(v'))$ as well.*

Proof. Proving $up(v) \sqsubseteq_{\mathcal{G}(q')} up(v')$ will prove $(q', up(v)) \sqsubseteq_{\mathcal{G}} (q', up(v'))$. Choose a constraint $\varphi \in \mathcal{G}(q')$. To prove this relation, it is sufficient (thanks to Proposition 3.3) to show that $up(v) \sqsubseteq_{\{\varphi\}} up(v')$. Since $(q, v) \sqsubseteq_{\mathcal{G}} (q, v')$, it means $v \sqsubseteq_{\mathcal{G}(q)} v'$. Now, since the constraint $up^{-1}(\varphi) \in \mathcal{G}(q)$, Proposition 3.2 implies $v \sqsubseteq_{\{up^{-1}(\varphi)\}} v'$. Then, Lemma 3.29 implies that $up(v) \sqsubseteq_{\{\varphi\}} up(v')$. □

The lemma above gives the final condition for the construction of $\mathcal{G}$:

$$\begin{array}{c} \textbf{for every transition } t = (q, g, R, q') \text{ and} \\ \textbf{for every constraint } \varphi \in \mathcal{G}(q'): \\ \mathcal{G}(q) \text{ contains the atomic constraint } up^{-1}(\varphi) \end{array} \tag{3.4}$$

Thanks to the lemma above, an $\mathcal{A}$-map can now be defined for Updatable Timed Automata, along the lines of Definition 3.22.

Definition 3.31 ($\mathcal{A}$-map in the presence of Updates)**.** *Let $\mathcal{A} = (Q, X, q_0, T, F)$ be a UTA. An $\mathcal{A}$-map $\mathcal{G}$ is a tuple $(\mathcal{G}(q))_{q \in Q}$ with each $\mathcal{G}(q)$ being a set of atomic constraints, such that the following conditions are satisfied. For every transition $(q, g, up, q') \in T$ the following holds:*

- *every atomic constraint of g belongs to $\mathcal{G}(q)$,*
- *$up^{-1}(0 \leq x) \in \mathcal{G}(q)$ for every $x \in X$,*
- *$up^{-1}(\varphi) \in \mathcal{G}(q)$ for every $\varphi \in \mathcal{G}(q')$* (*called the* propagation criterion)

An $\mathcal{A}$-map of Definition 3.31 can be computed by modifying Algorithm 2: (i) Line 5 should be changed to the following:

$$\mathcal{G}(q) \leftarrow \mathcal{G}(q) \cup \mathsf{atoms\_of}(g) \cup \{up^{-1}(0 \leq x) \mid x \in X\}$$

and (ii) Line 9 should be changed to the following:

$$\mathcal{G}(q) \leftarrow \mathcal{G}(q) \cup \{up^{-1}(\varphi)\}$$

However, such $\mathcal{A}$-map may not result in minimal $\mathcal{G}(q)$ sets. Although it is sufficient (as proved in Lemma 3.30), it is *not always necessary* for $\mathcal{G}(q)$ to contain $up^{-1}(\varphi)$ in order to satisfy the condition: if $(q, v) \sqsubseteq_{\mathcal{G}} (q, v')$ then $(q', up(v)) \sqsubseteq_{\mathcal{G}} (q', up(v'))$ for every transition (q, g, up, q'). This realization stems from the fact that the reverse directions of Lemma 3.20 and 3.29 do not hold.

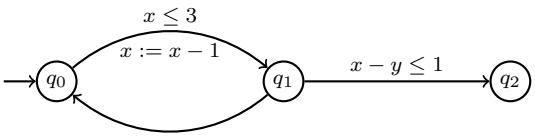


Figure 3.3: Non-terminating $\mathcal{A}$-map computation

Consider the automaton in Figure 3.3. The computation of Algorithm 2 (with the two modifications described on Page 46) on this automaton is depicted in Figure 3.4. Clearly, this $\mathcal{A}$-map computation does not terminate.

$$\begin{array}{l} \mathcal{G}(q_0) = \{\} \\ \mathcal{G}(q_1) = \{\} \\ \mathcal{G}(q_2) = \{\} \end{array} \longrightarrow \begin{array}{l} \{x \leq 3, 1 \leq x\} \\ \{x - y \leq 1\} \\ \{\} \end{array} \longrightarrow \begin{array}{l} \{x \leq 3, 1 \leq x, x - y \leq 2\} \\ \{x \leq 3, 1 \leq x, x - y \leq 1\} \\ \{\} \end{array} \longrightarrow \begin{array}{l} \{x \leq 3, x \leq 4, 1 \leq x, 2 \leq x, x - y \leq 2\} \\ \{x \leq 3, 1 \leq x, x - y \leq 1, x - y \leq 2\} \\ \{\} \end{array}$$

$$\swarrow$$

$$\cdots \longleftarrow \begin{array}{l} \{x \leq 3, x \leq 4, 1 \leq x, 2 \leq x, x - y \leq 2, x - y \leq 3\} \\ \{x \leq 3, x \leq 4, 1 \leq x, 2 \leq x, x - y \leq 1, x - y \leq 2\} \\ \{\} \end{array}$$

Figure 3.4: $\mathcal{A}$-map computation for the automaton in Figure 3.3; at every step the new constraints that get added are marked in red

The first step adds constraints that meet the first two conditions of Definition 3.31. Since for the updates present in each of the transitions of the automaton, $up^{-1}(0 \leq y)$ is $0 \leq y$, which is semantically equivalent to $\top$, it is not added explicitly to either $\mathcal{G}(q_0)$ or $\mathcal{G}(q_1)$. Consider two transitions $(q_0, v) \xrightarrow{t} (q_1, up(v))$ and $(q_0, v') \xrightarrow{t} (q_1, up(v'))$ with $t = (q_0,\ x \leq 3,\ x := x - 1,\ q_1)$, and up being $x := x - 1$. Then, $(up(v))(x) = v(x) - 1$ and $(up(v'))(x) = v'(x) - 1$. Suppose, it is required that $up(v) \sqsubseteq_{\{x-y<1\}} up(v')$. By Definition 3.1, the condition that needs to be satisfied is: if $up(v) \models x - y < 1$, then $up(v') \models x - y < 1$. Rewriting in terms of v: if $v(x) - 1 - v(y) < 1$, then $v'(x) - 1 - v'(y) < 1$. In other words: if $v \models x - y < 2$, then $v' \models x - y < 2$. This is achieved by adding $x - y < 2$, that is the constraint $up^{-1}(x - y < 1)$, to $\mathcal{G}(q_0)$ in the second step. This is the essence of the propagation criterion of Definition 3.31, which asks that for each $\varphi \in \mathcal{G}(q_1)$, the set $\mathcal{G}(q_0)$ contains $up^{-1}(\varphi)$. The fixpoint computation iteratively ensures this criterion for each transition of the automaton. As illustrated in Figure 3.4, the computation does not terminate for the automaton of Figure 3.3. There are three sources of increasing constants: (1) $x \leq 3, x \leq 4, \ldots$, (2) $1 \leq x, 2 \leq x, \ldots$ and (3) $x - y < 1, x - y < 2, \ldots$.

The claim now is: this conservative propagation is unnecessary to get the required simulation. Suppose $v \sqsubseteq_{\mathcal{G}(q_0)} v'$ and $(q_0, v) \xrightarrow{t} (q_1, up(v))$, with $t := (q_0, x \leq 3, x := x - 1, q_1)$. Since t is enabled at v, it means $v(x) \leq 3$ and hence $v'(x) \leq v(x) \leq 3$ since the atomic constraint $x \leq 3$ is present in $\mathcal{G}(q_0)$. Then, it

follows that both the valuations v and v' satisfy the constraint $x - y \leq 3$, since every valuation satisfies $y \geq 0$. The presence of $x - y < 4,\ x - y < 5,\ \ldots$ at $\mathcal{G}(q_0)$ is useless as both v, v' already satisfy these constraints. Stopping the propagation of $x - y < 3$ from $\mathcal{G}(q_1)$ will cut the infinite propagation due to (3). Similarly, v and v' both satisfy the constraints $x \leq 3, x \leq 4, \ldots$, therefore the propagation of $x \leq 3$ from $\mathcal{G}(q_1)$ (that is, adding the constraint $x \leq 4$) to $\mathcal{G}(q_0)$ is not necessary, stopping (1). The remaining source (2) is trickier, but it can still be eliminated. Here is the main idea. Consider a constraint $3 \leq x \in \mathcal{G}(q_0)$ which propagates unchanged to $\mathcal{G}(q_1)$ and then back to $\mathcal{G}(q_0)$ as $up^{-1}(3 \leq x) = 4 \leq x$. This propagation can be cut since $v \sqsubseteq_{\{3 \leq x\}} v'$ already ensures $v \sqsubseteq_{\{4 \leq x\}} v'$ for the valuations that are *relevant*: the ones that satisfy the guard $x \leq 3$ of t. Indeed, $v, v' \models x \leq 3$ and $v \sqsubseteq_{\{3 \leq x\}} v'$ implies $v(x) \leq v'(x)$ which in turn implies $v \sqsubseteq_{\{4 \leq x\}} v'$. Overall, it can be shown that the $\mathcal{A}$-map $\mathcal{G}$ with $\mathcal{G}(q_0) = \{x \leq 3,\ 3 \leq x,\ x - y < 2,\ x - y < 3\}$ and $\mathcal{G}(q_1) = \{x - y < 1\} \cup \mathcal{G}(q_0)$ suffices to make the relation $\sqsubseteq_{\mathcal{G}}$ a simulation relation.

Taking guards into account for propagations. The propagation criterion of Definition 3.31, which is responsible for non-termination, is oblivious to the guard that is present in the transition. A new propagation criterion is proposed below that takes the guard into account and cuts out certain irrelevant constraints. Consider a transition (q, g, up, q') and a constraint $\varphi \in \mathcal{G}(q')$. What actually is required is a constraint $\psi \in \mathcal{G}(q)$ such that $v \sqsubseteq_{\{\psi\}} v'$ *and* $v \models g$ and $v \sqsubseteq_{\mathsf{atoms\_of}(g)} v'$ implies $up(v) \sqsubseteq_{\{\varphi\}} up(v')$. The additional "and $v \models g$ and $v \sqsubseteq_{\mathsf{atoms\_of}(g)} v'$ " was missing in the intuition behind the previous propagation. Of course, setting $\psi := up^{-1}(\varphi)$ is sufficient - even without the additional information - as proved in Lemma 3.29. However, the goal is to either eliminate the need for ψ or find a ψ with a smaller constant compared to $up^{-1}(\varphi)$. It turns out that in fact in many cases, it is possible to get the former, when the additional condition "and $v \models g$" is plugged in.

Definition 3.32 (pre under a "guard-update" pair)**.** *Given an atomic constraint φ and a "guard-update" pair (g, up), the* pre *of the constraint φ under this pair, written as* $\mathsf{pre}(\varphi, g, up)$ *is: an atomic constraint as specified in Table 3.1 if g and $up^{-1}(\varphi)$ satisfy the corresponding conditions mentioned in the table, otherwise* $\mathsf{pre}(\varphi, g, up)$ *is the atomic constraint $up^{-1}(\varphi)$.*

	$up^{-1}(\varphi)$	if g contains	$\mathsf{pre}(\varphi, g, up)$
1	$x \lhd d$	$x \lhd_1 c$	$\top$
2	$d \lhd x$	$x \lhd_1 c$ with $c < d$	$c \leq x$
3	$x - y \lhd d$ or $d \lhd x - y$	$x \lhd_1 c$ or $x - y \lhd_1 c$ or $e \lhd_1 x - y$ s.t. $c < d < e$	$\top$

Table 3.1: Cases where pre under "guard-update" pair is different from $up^{-1}(\varphi)$. Two different symbols $\lhd$ and $\lhd_1$ are used in the constraints above to insist that the inequality $\lhd$ need not be same as the inequality $\lhd_1$.

The next lemma states: given a transition (q, g, up, q') and an atomic constraint φ, it is sufficient that the relation $v \sqsubseteq_{\{\mathsf{pre}(\varphi, g, up)\}} v'$ holds in order to ensure that

the updated valuations satisfy the relation $up(v) \sqsubseteq_{\{\varphi\}} up(v')$, when the "additional information" regarding the guard is taken into account.

Lemma 3.33. *Let (g, up) be a guard-update pair, φ be an atomic constraint and v, v' be two valuations such that $up(v) \geq 0$, $up(v') \geq 0$, $v \models g$ and $v \sqsubseteq_{\mathsf{atoms\_of}(g)} v'$. Then, if the relation $v \sqsubseteq_{\{\mathsf{pre}(\varphi,g,up)\}} v'$ holds then $up(v) \sqsubseteq_{\{\varphi\}} up(v')$ holds as well.*

Proof. When $\mathsf{pre}(\varphi, g, up) = up^{-1}(\varphi)$, this is Lemma 3.29. First it is proved that the relation $v \sqsubseteq_{\{up^{-1}(\varphi)\}} v'$ holds for the combinations given in Table 3.1.

(*Case 1*) From the hypothesis $v \sqsubseteq_{\{\varphi \mid \varphi \in \mathsf{atoms\_of}(g)\}} v'$, since the guard g contains the atomic constraint $x \lhd_1 c$, that is, $x \lhd_1 c \in \mathsf{atoms\_of}(g)$, it follows from Theorem 3.2 that $v \sqsubseteq_{\{x \lhd_1 c\}} v'$. Theorem 3.5 then implies that either $v \not\models x \lhd_1 c$ or $v'(x) \leq v(x)$. From the other hypothesis $v \models g$, it follows that $v \models x \lhd_1 c$. Therefore, the inequality $v'(x) \leq v(x)$ holds. Theorem 3.5 then implies that $v \sqsubseteq_{\{x \lhd d\}} v'$ holds for every upper bounded non-diagonal constraint $x \lhd d$.

(*Case 2*) In this case, $\mathsf{pre}(\varphi, g, up) = c \leq x$ with $c < d$. Since the guard g contains the atomic constraint $x \lhd_1 c$, from the reasoning given in the previous case, it follows that $v'(x) \leq v(x)$. Since v satisfies the guard, it follows that $v'(x) \leq v(x) \lhd_1 c < d$. Now, if $v \sqsubseteq_{\{\mathsf{pre}(\varphi,g,up)\}} v'$, that is, $v \sqsubseteq_{\{c \leq x\}} v'$, from Theorem 3.9, it follows that: either $c \leq v'(x)$ or $v(x) \leq v'(x)$. Using $v'(x) \leq v(x) \lhd_1 c$, it follows that, in both cases, $v(x) = v'(x)$. Hence $v \sqsubseteq_{\{d \lhd x\}} v'$.

(*Case 3*) There are sub-cases depending on whether the guard contains a non-diagonal constraint or the diagonal constraints. To prove that $v \sqsubseteq_{\{x-y \lhd d\}} v'$ and $v \sqsubseteq_{\{d \lhd x-y\}} v'$. When the guard contains $x \lhd_1 c$, it follows that $v'(x) \leq v(x) \lhd_1 c$ as above. Hence, $v'(x) - v'(y) \lhd_1 c$ and $v(x) - v(y) \lhd_1 c$, since $v(y) \geq 0$ and $v'(y) \geq 0$. Since it is given that $c < d$, both v and v' satisfy the diagonal constraint $x - y \lhd d$ and neither of them satisfies $d \lhd x - y$. Notice that, time elapse preserves the satisfaction of diagonal constraints, as for every valuation u and every constant $\delta \in \mathbb{R}_{\geq 0}$, $(u + \delta)(x) - (u + \delta)(y) = u(x) - u(y)$. From Definition 3.1, $v \sqsubseteq_{\{\psi\}} v'$ for a diagonal constraint ψ is satisfied, if $v \not\models \psi$ or $v' \models \psi$. Hence, since $v' \models x - y \lhd d$, $v \sqsubseteq_{\{x-y \lhd d\}} v'$ holds and since $v \not\models d \lhd x - y$, $v \sqsubseteq_{\{d \lhd x-y\}} v'$ holds.

For the other sub-cases of the guard containing $x - y \lhd_1 c$ or $e \lhd_1 x - y$, the hypotheses $v \models g$, $v \sqsubseteq_{\mathsf{atoms\_of}(g)} v'$ and the fact that $c < d < e$ ensure the same effect, that either v does not satisfy the diagonal constraint $up^{-1}(\varphi)$ or v' does. Hence, from Definition 3.1 the relation $v \sqsubseteq_{\{up^{-1}(\varphi)\}} v'$ holds.

Therefore, in all the cases above, the relation $v \sqsubseteq_{\{up^{-1}(\varphi)\}} v'$ holds. Then, Lemma 3.29 implies that $up(v) \sqsubseteq_{\{\varphi\}} up(v')$. This completes the proof. □

Thanks to the lemma above, a new $\mathcal{A}$-map can now be defined that uses the constraint $\mathsf{pre}(\varphi, g, up)$ instead of $up^{-1}(\varphi)$. Since in some cases the constraint $\mathsf{pre}(\varphi, g, up)$ is $\top$ whereas the constraint $up^{-1}(\varphi)$ is non-trivial, this new $\mathcal{A}$-map can potentially contain smaller $\mathcal{G}(q)$ sets.

Definition 3.34 (*reduced* $\mathcal{A}$-map)**.** *A* reduced *$\mathcal{A}$-map is a tuple $\mathcal{G} := (\mathcal{G}(q))_{q \in Q_\mathcal{A}}$, where each $\mathcal{G}(q)$ is a set of atomic constraints and satisfying the following two conditions for every state q and for every transition $t = (q, g, up, q_1)$ of $\mathcal{A}$:*

– *$\mathcal{G}(q)$ contains the atomic constraints present in the guard g,*

– $\mathcal{G}(q)$ *contains the atomic constraint* $\mathsf{pre}(0 \leq x, g, up)$,

– *for every* $\varphi \in \mathcal{G}(q_1)$, $\mathcal{G}(q)$ *contains the atomic constraint* $\mathsf{pre}(\varphi, g, up)$.

Like the $\mathcal{A}$-map of Definition 3.22, the *reduced* $\mathcal{A}$-map can also be obtained as the least fixpoint of a system of equations as mentioned below.

Lemma 3.35. *The smallest reduced $\mathcal{A}$-map with respect to pointwise inclusion is the least fixpoint of the following system of equations:*

$$\mathcal{G}(q) = \bigcup_{(q,g,up,q_1)} \{\mathsf{atoms\_of(g)} \cup \{\mathsf{pre}(0 \leq x, g, up) \mid x \in X\} \cup \{\mathsf{pre}(\varphi, g, up) \mid \varphi \in \mathcal{G}(q_1)\}\}$$

Proof. Every reduced $\mathcal{A}$-map is a fixpoint of the given system of equations and every solution to the given system of equations is a reduced $\mathcal{A}$-map. □

The following theorem proves that the relation $\sqsubseteq_{\mathcal{G}}$, when $\mathcal{G}$ is a reduced $\mathcal{A}$-map, satisfies the conditions of a simulation relation.

Theorem 3.36. *Given an updatable timed automaton $\mathcal{A} = (Q, X, q_0, T, F)$, the relation $\sqsubseteq_{\mathcal{G}}$ of Definition 3.15 defined over the space of configurations of $\mathcal{A}$ is a simulation relation if the tuple $(\mathcal{G}(q))_{q \in Q}$ is a reduced $\mathcal{A}$-map.*

Proof. The relation $\sqsubseteq_{\mathcal{G}}$ needs to satisfy the conditions 1, 2a, 2b and 2c given on Page 44 in order to become a simulation relation. Lemma 3.17 ensures that $\sqsubseteq_{\mathcal{G}}$ satisfies condition 1, Lemma 3.18 ensures $\sqsubseteq_{\mathcal{G}}$ satisfies condition 2a and Lemma 3.27 ensures $\sqsubseteq_{\mathcal{G}}$ satisfies condition 2b. Finally, Lemma 3.33 implies that $\sqsubseteq_{\mathcal{G}}$ satisfies the condition 2c as well. Therefore, if $\mathcal{G}$ is a reduced $\mathcal{A}$-map then the relation $\sqsubseteq_{\mathcal{G}}$ is a simulation relation for Updatable Timed Automata. □

Given an updatable timed automaton $\mathcal{A}$, the reduced $\mathcal{A}$-map can be computed using Algorithm 3, which is obtained through a little modification to Algorithm 2.

Unlike Algorithm 2, that terminates on every input timed automaton, Algorithm 3 does not necessarily terminate given an updatable timed automaton as input. Given an input UTA, determining if Algorithm 3 will terminate or not, is possible. This is the topic of the next section.

3.4 Termination of parameter computation

In order to use the simulation relation $\sqsubseteq_{\mathcal{G}}$ in a reachability algorithm, an appropriate parameter $\mathcal{G}$ needs to be computed first. Since the reduced $\mathcal{A}$-map computes the smallest $\mathcal{G}(q)$ sets among the constructions presented in the previous section, this is the parameter that will be used in the simulation relation.

As explained on Page 47, the construction of $\mathcal{A}$-map with $\mathsf{pre}(\varphi, up)$ (Definition 3.28) does not terminate for the automaton in Figure 3.3. The construction of *reduced* $\mathcal{A}$-map (Definition 3.34) based on the function $\mathsf{pre}(\varphi, g, up)$ (Definition 3.32) terminates for this example automaton. However, given an updatable timed automaton, the computation of reduced $\mathcal{A}$-map is also not guaranteed to terminate

Input: $\mathcal{A} = (Q, X, q_0, T, F)$
Output: $\mathcal{G}(q)$ for every state $q \in Q$

foreach *state* $q \in Q$ **do**
  $\mathcal{G}(q) \leftarrow \emptyset$
end
foreach *transition* $t = (q, g, up, q') \in T$ **do**
  $\mathcal{G}(q) \leftarrow \mathcal{G}(q) \cup \mathsf{atoms\_of}(g) \cup \{\mathsf{pre}(0 \leq x, g, up) \mid \forall x \in X\}$
end
while *a fixpoint of* $(\mathcal{G}(q))_{q \in Q}$ *is not reached* **do**
  foreach *transition* $t = (q, g, up, q') \in T$ **do**
    foreach *constraint* φ *in* $\mathcal{G}(q')$ **do**
      $\mathcal{G}(q) \leftarrow \mathcal{G}(q) \cup \{\mathsf{pre}(\varphi, g, up)\}$
    end
  end
end

Algorithm 3: Computing reduced $\mathcal{A}$-map

always. It is therefore important to know, given an updatable timed automaton as input, whether the reduced $\mathcal{A}$-map computation will terminate for the input or not.

Consider the automaton in Figure 3.5. Note that, according to Table 3.1, the constraint $\mathsf{pre}(x - y \leq i,\ 2 < x,\ x := x - 1) = x - y \leq i + 1$, for every integer $i \geq 1$. Therefore, the reduced $\mathcal{A}$-map will be the tuple $(\mathcal{G}(q_0), \mathcal{G}(q_1))$, where -

$$\mathcal{G}(q_0) = \{x - y \leq i \mid i \geq 1,\ i \in \mathbb{Z}\}$$
$$\mathcal{G}(q_1) = \{\}$$

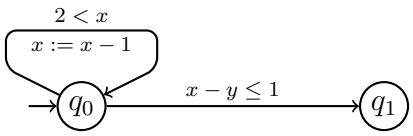


Figure 3.5: Automaton for which reduced $\mathcal{A}$-map computation does not terminate

Since the set $\mathcal{G}(q_0)$ is infinite, the reduced $\mathcal{A}$-map computation will not terminate for this automaton. It is, however, possible to check if the reduced $\mathcal{A}$-map computation is going to terminate, given an updatable timed automaton as input. The rest of this section discusses how to check this. The results presented in this section are part of the work [GMS20].

Let $\mathcal{A} = (Q, X, q_0, T, F)$ be an updatable timed automaton with:

$$M = \max\{c \mid c \text{ occurs in some guard of } \mathcal{A}\}$$
$$L = \max\{|d| \mid d \text{ occurs in some update of } \mathcal{A}\}$$

Let $\mathcal{G}$ be the smallest reduced $\mathcal{A}$-map computed by the least fixpoint of the equations in Lemma 3.35. The claim is: this fixpoint computation does not terminate if a constraint with a large enough constant gets added to some $\mathcal{G}(q)$.

Proposition 3.37. *The reduced $\mathcal{A}$-map computation does not terminate iff for some state q, there is an atomic constraint $\varphi \in \mathcal{G}(q)$ with a constant $c_\varphi > N$, where $N = \max(M, L) + (2 \cdot L \cdot |Q| \cdot |X|^2)$.*

For the analysis, the strings of the form "$x \leq$", "$\leq x$", "$x - y \leq$" and "$\leq x - y$" where $x, y \in X$, will be used and will be called *contexts*. Given such a context $\overline{\varphi}$ and a constant c, $\overline{\varphi}[c]$ denotes the atomic constraint obtained by plugging the constant into the context. For example, if $\overline{\varphi} = \leq x$ and $c = 2$ then $\overline{\varphi}[c]$ is $2 \leq x$.

In the proof below, the notion of *propagation sequence* will be used, which is a sequence of the form $(q_i, \overline{\varphi}_i[c_i]) \to (q_{i+1}, \overline{\varphi}_{i+1}[c_{i+1}]) \to \cdots \to (q_j, \overline{\varphi}_j[c_j])$ such that for all $i \leq k < j$, the atomic constraint $\overline{\varphi}_{k+1}[c_{k+1}] = \mathsf{pre}(\overline{\varphi}_k[c_k], g_k, up_k)$ for some transition $(q_{k+1}, g_k, up_k, q_k)$ of the given updatable timed automaton $\mathcal{A}$.

Proof (of Proposition 3.37). Since the constants appearing in an atomic constraint are always non-negative, if the reduced $\mathcal{A}$-map computation does not terminate, there must be constraints with growing constants being added to the sets $\mathcal{G}(q)$'s. The left to right implication of Proposition 3.37 is therefore clear. Conversely, assume that $\overline{\varphi}[c] \in \mathcal{G}(q)$, where the constant $c > \max(M, L) + 2 \cdot L \cdot |Q| \cdot |X|^2$. Consider the smallest $n \geq 0$ such that $\overline{\varphi}[c] \in \mathcal{G}^n(q)$. Then, there is a propagation sequence $\pi = (q_i, \overline{\varphi}_0[c_0]) \to (q_{i+1}, \overline{\varphi}_1[c_1]) \to \cdots \to (q_{i+n}, \overline{\varphi}_n[c_n])$ such that $\overline{\varphi}_0[c_0] \in \mathcal{G}^0(q_i)$ and $(q_{i+n}, \overline{\varphi}_n[c_n]) = (q, \overline{\varphi}[c])$. Note that $\overline{\varphi}_j[c_j] \in \mathcal{G}^j(q_{i+j})$ for all $0 \leq j \leq n$. First claim: the propagation sequence π contains a positive cycle with *large* constants.

Lemma 3.38. *There exists $0 < j_1 < j_2 \leq n$ such that $(q_{i+j_1}, \overline{\varphi}_{j_1}) = (q_{i+j_2}, \overline{\varphi}_{j_2})$, $c_{j_1} < c_{j_2}$ and $\max(M, L) < c_k$ for all $j_1 \leq k \leq j_2$.*

Proof. Since $\overline{\varphi}_0[c_0] \in \mathcal{G}^0(q_i)$, the atomic constraint $\overline{\varphi}_0[c_0]$ is either present in a guard or it is $\mathsf{pre}(0 \leq x, g, up)$ for some clock x and a guard-update pair (g, up) which is present in some transition. Therefore, c_0 satisfies the relation $0 \leq c_0 \leq \max(M, L)$. Consider the last occurrence of a *small* constant in the propagation sequence. More precisely, let $m = \max\{k \mid 0 \leq k < n \text{ and } c_k \leq \max(M, L)\}$. Hence, for all $m < k \leq n$, the constant c_k is such that $c_k > \max(M, L)$.

Notice that, for $m < k < n$, the constraint in the sequence cannot switch from an upper diagonal to a lower diagonal and vice-versa. This is because, if $\overline{\varphi}_k[c_k] = (x - y \leq c_k)$ and $\overline{\varphi}_{k+1}[c_{k+1}] = (c_{k+1} \leq y' - x')$, then the update up_k must contain $x := x' + d$, $y := y' + e$ with $c_{k+1} = d - e - c_k$. This is not possible with $|d|, |e| \leq L$ and $c_k, c_{k+1} > L$. Similarly, it can be shown that an upper (resp. lower) diagonal constraint cannot switch to a lower (resp. upper) non-diagonal constraint. On the other hand, it is possible to switch once from an upper (resp. lower) diagonal constraint to an upper (resp. lower) non-diagonal constraint.

The other remark is: $|c_{k+1} - c_k| \leq 2L$ for all $m \leq k < n$. Since $c_m \leq \max(M, L)$ and $c_n > \max(M, L) + 2 \cdot L \cdot |Q| \cdot |X|^2$, there exists an increasing sequence $m < i_1 < i_2 < \cdots < i_\ell \leq n$ with $c_{i_1} < c_{i_2} < \cdots < c_{i_\ell}$ and $\ell > |Q| \cdot |X|^2$. As noticed above, the contexts $\overline{\varphi}_k$ are either all upper constraints or all lower constraints, hence the set $\{(q_{i+k}, \overline{\varphi}_k) \mid m < k \leq n\}$ contains at most $|Q||X|^2$ elements (for every state, there are at most $|X|$ many non-diagonals and $|X|(|X| - 1)$ many diagonals). Therefore, there exists $j_1, j_2 \in \{i_1, \ldots, i_\ell\}$ such that $j_1 < j_2$, hence $c_{j_1} < c_{j_2}$, and

$(q_{i+j_1}, \overline{\varphi}_{j_1}) = (q_{i+j_2}, \overline{\varphi}_{j_2})$. Also, recall that $c_k > \max(M, L)$ for all $m < k \leq n$ and $m < j_1 < j_2 \leq n$. □

The next step is to show that a positive cycle with large constants can be iterated, resulting in larger and larger constants.

Lemma 3.39. *Let* $(q_i, \overline{\varphi}_i[c_i]) \to (q_{i+1}, \overline{\varphi}_{i+1}[c_{i+1}]) \to \cdots \to (q_j, \overline{\varphi}_j[c_j])$ *be a propagation sequence with* $(q_i, \overline{\varphi}_i) = (q_j, \overline{\varphi}_j)$, $d = c_j - c_i > 0$ *and* $c_k > M$ *for all* $i \leq k \leq j$. *Then,* $(q_i, \overline{\varphi}_i[c_i + d]) \to (q_{i+1}, \overline{\varphi}_{i+1}[c_{i+1} + d]) \to \cdots \to (q_j, \overline{\varphi}_j[c_j + d])$ *is also a propagation sequence.*

Proof. Let $i \leq k < j$ and $(q_{k+1}, g_k, up_k, q_k)$ be a transition of $\mathcal{A}$ yielding the propagation from k to $k+1$: $\overline{\varphi}_{k+1}[c_{k+1}] = \mathsf{pre}(\overline{\varphi}_k[c_k], g_k, up_k)$. Since $c_{k+1} > M$, none of the three cases of Table 3.1 applies: if $\overline{\varphi}_{k+1}$ is one of $x \leq$, $\leq x$, $x - y \leq$ or $\leq x - y$ then g_k does not contain $x \leq c$ or $x - y \leq c$. Hence, the constraint $\overline{\varphi}_{k+1}[c_{k+1}] = up^{-1}(\overline{\varphi}_k[c_k])$. From the definition of up^{-1} (in Definition 3.28), it can be deduced that $\overline{\varphi}_{k+1}[c_{k+1} + d] = up^{-1}(\overline{\varphi}_k[c_k + d])$. Since $c_{k+1} + d > M$ the cases of Table 3.1 do not apply as well and hence $\overline{\varphi}_{k+1}[c_{k+1}+d] = \mathsf{pre}(\overline{\varphi}_k[c_k+d], g_k, up_k)$. □

This allows to conclude the proof of Proposition 3.37. Using Lemma 3.38, a positive cycle with *large* constants can be obtained. This cycle can then be iterated forever thanks to Lemma 3.39. Therefore, $\overline{\varphi}_i[c_i + kd] \in \mathcal{G}^i(q_i)$ for all $k \geq 0$ and the reduced $\mathcal{A}$-map computation does not terminate. □

Algorithm for Deciding Termination. Proposition 3.37 gives a termination mechanism: run the fixpoint computation $\mathcal{G}^0, \mathcal{G}^1, \ldots$ – stop if either the computation stabilizes with $\mathcal{G}^n = \mathcal{G}^{n+1}$ or if some constraint $\varphi(:= \overline{\varphi}[c_\varphi])$ gets added to $\mathcal{G}^n(q)$ with $c_\varphi > N$, where $N = \max(M, L) + (2 \cdot L \cdot |Q| \cdot |X|^2)$. This is formalized in Algorithm 4.

The number of pairs (q, φ) with $c_\varphi \leq N$ is $2 \cdot N \cdot |Q| \cdot |X|^2$ (the factor 2 is for upper or lower constraints). Therefore, Algorithm 4 stops after at most $2 \cdot N \cdot |Q| \cdot |X|^2$ steps and the total computation time is $\mathsf{poly}(M, L, |Q|, |X|)$.

If the constants occurring in guards and updates of the UTA $\mathcal{A}$ are encoded in unary, the computation of $\mathcal{G}$ terminates in time $\mathsf{poly}(|\mathcal{A}|)$. If the constants are encoded in binary, (non-)termination of the computation of $\mathcal{G}$ can be detected in NPSPACE = PSPACE: it suffices to search for a propagation sequence $(q_i, \varphi_0) \to (q_{i+1}, \varphi_1) \to \cdots \to (q_{i+n}, \varphi_n)$ such that $\varphi_0 \in \mathcal{G}^0(q_i)$ and $c_{\varphi_n} > N$. For this, only the current pair (q_{i+k}, φ_k) needs to be stored, guess a transition $(q_{i+k+1}, g_k, up_k, q_{i+k})$ of $\mathcal{A}$ and compute the next pair $(q_{i+k+1}, \varphi_{k+1})$ with $\varphi_{k+1} = \mathsf{pre}(\varphi_k, g_k, up_k)$. This can be done with polynomial space. A matching PSPACE lower-bound is shown below.

Lower bound. A reduction from the control-state reachability of bounded one-counter automata is shown to prove that when constants are encoded in binary, deciding termination of the reduced propagation is PSPACE-hard.

A *bounded one-counter automaton* [FJ15, HOW16] is given by (L, ℓ_0, Δ, b) where L is a finite set of states, ℓ_0 is an initial state, Δ is a set of transitions and $b \geq 0$ is the global bound for the counter. Each transition is of the form (ℓ, p, ℓ') where ℓ is the source and ℓ' the target state of the transition, $p \in [-b, +b]$ gives the update to the counter. A run of the counter automaton is a sequence $(\ell_0, c_0) \to (\ell_1, c_1) \to$

Input: an updatable timed automaton $\mathcal{A} = (Q, X, q_0, T, F)$
Output: $\mathcal{G}(q)$ for every state $q \in Q$

foreach *state* $q \in Q$ **do**
  $\mathcal{G}(q) \leftarrow \emptyset$
end
$N \leftarrow \max(M, L) + (2 \cdot L \cdot |Q| \cdot |X|^2)$;
foreach *transition* $t = (q, g, up, q') \in T$ **do**
  $\mathcal{G}(q) \leftarrow \mathcal{G}(q) \cup \mathsf{atoms\_of}(g) \cup \{\mathsf{pre}(0 \leq x, g, up) \mid \forall x \in X\}$
end
while *a fixpoint of* $(\mathcal{G}(q))_{q \in Q}$ *is not reached* **do**
  foreach *transition* $t = (q, g, up, q') \in T$ **do**
    foreach *constraint* φ *in* $\mathcal{G}(q')$ **do**
      $\overline{\varphi}[c_\varphi] \leftarrow \mathsf{pre}(\varphi, g, up)$;
      if $c_\varphi \leq N$ **then**
        $\mathcal{G}(q) \leftarrow \mathcal{G}(q) \cup \{\overline{\varphi}[c_\varphi]\}$; // add $\mathsf{pre}(\varphi, g, up)$ to $\mathcal{G}(q)$
      else
        conclude that the fixpoint computation will not terminate ;
        return ;
      end
    end
  end
end

Algorithm 4: Reduced $\mathcal{A}$-map computation with termination check

$\cdots \to (\ell_n, c_n)$ such that $c_0 = 0$, each $c_i \in [0, b]$ and there are transitions $(\ell_i, p_i, \ell_{i+1})$ with $c_{i+1} = c_i + p_i$. All constants used in the automaton definition are encoded in binary. Reachability problem for this model asks: does there exist a run starting from $(\ell_0, 0)$ to a given state ℓ_t with some (arbitrary) counter value c_t. This problem is known to be PSPACE-complete [FJ15]. Now, a reduction will be given from the reachability for bounded one-counter automata to the problem of checking whether the fixpoint computation for the smallest reduced $\mathcal{A}$-map terminates.

From a bounded one counter automaton $\mathcal{B} = (L, \ell_0, \Delta, b)$, construct a UTA $\mathcal{A}_\mathcal{B}$ in the following way. States of $\mathcal{A}_\mathcal{B}$ are $L \cup \{\ell'_0, \ell'_t\}$ where ℓ'_0 and ℓ'_t are *new* states *not in* L. $\mathcal{A}_\mathcal{B}$ contains two clocks x, y. For each transition (ℓ, p, ℓ') of $\mathcal{B}$, there is a transition (ℓ', g, up, ℓ) in $\mathcal{A}_\mathcal{B}$, with the guard g being $x \leq b \wedge y \leq 0$ and the update up being $x := x - p$ and $y := y$. There are three additional transitions using the new states ℓ'_0 and ℓ'_t: (1) $\ell_0 \xrightarrow{x-y \leq 0} \ell'_0$, (2) $\ell'_t \to \ell_t$ and (3) $\ell'_t \xrightarrow{x:=x, y:=y+1} \ell'_t$.

In the reduced $\mathcal{A}_\mathcal{B}$-map computation, the constraint $x - y \leq 0$ is added to $\mathcal{G}^0(\ell_0)$. The propagation sequence starting from $(\ell_0, x - y \leq 0)$ mimics the runs of the counter machine $\mathcal{B}$ with the constant present in the diagonal constraint $x - y \leq c$ giving the value of the counter. Case 3 of Table 3.1 keeps this value bounded between 0 and b. Guard $x \leq b$ disallows propagation of constraints $x - y \leq d$ with $d > b$. But, it can allow d to go smaller and smaller, and at one point the constant becomes negative and the constraint gets rewritten: $x - y \leq b, x - y \leq b - 1, \ldots, x - y \leq$

$0, 1 \leq y - x, 2 \leq y - x$, etc. The presence of the constraint $y \leq 0$ in the guard eliminates $1 \leq y - x, 2 \leq y - x$, etc. once again due to Case 3 of Table 3.1.

Lemma 3.40. *For every run $(\ell_0, 0) \to (\ell_1, c_1) \to \cdots \to (\ell_n, c_n)$ in the bounded one-counter automaton $\mathcal{B}$, there exists a propagation sequence $(\ell_0, x - y \leq 0) \to (\ell_1, x - y \leq c_1) \to \cdots \to (\ell_n, x - y \leq c_n)$ in the updatable timed automaton $\mathcal{A}_\mathcal{B}$.*

Proof. Let $\mathcal{G}$ be the smallest reduced $\mathcal{A}_\mathcal{B}$-map, that is, for every reduced $\mathcal{A}_\mathcal{B}$-map $\mathcal{G}'$ and for every state q of $\mathcal{A}_\mathcal{B}$, $\mathcal{G}(q) \subseteq \mathcal{G}'(q)$ holds. It will be shown by induction, that, for every i, there is a constraint $x - y \leq c_i$ in $\mathcal{G}(\ell_i)$. Due to the edge $\ell_0 \xrightarrow{x-y\leq 0} \ell'_0$, the constraint $x - y \leq 0 \in \mathcal{G}(\ell_0)$. Suppose, the hypothesis is true for some $j > 0$, that is, there exists $x - y \leq c_j \in \mathcal{G}(\ell_j)$. Since the run contains $(\ell_j, c_j) \to (\ell_{j+1}, c_{j+1})$, there is a transition (ℓ_j, p, ℓ_{j+1}) in $\mathcal{B}$ and $c_{j+1} = c_j + p$ with $0 \leq c_{j+1} \leq b$. By construction, there is a transition $(\ell_{j+1}, g, up, \ell_j)$ in $\mathcal{A}_\mathcal{B}$ with $up_x = x - p$ and guard $x \leq b \wedge y \leq 0$. Hence the constraint $\varphi = x - y \leq c_j$ at $\mathcal{G}(\ell_j)$ should be propagated to $\mathcal{G}(\ell_{j+1})$. The constraint $up^{-1}(\varphi) = x - y \leq c_j + p$, that is $x - y \leq c_{j+1}$. Since $0 \leq c_{j+1} \leq b$, Case 3 of Table 3.1 does not apply: the constraint $x \leq b$ in g does not cut the propagation since $c_{j+1} \leq b$, and the constraint $y \leq 0$ in g does not apply as well (if $c_{j+1} = 0$ the constraint $up^{-1}(\varphi)$ can also be written as $0 \leq y - x$). Therefore, $\mathsf{pre}(x - y \leq c_j, g, up) = x - y \leq c_{j+1} \in \mathcal{G}(\ell_{j+1})$. □

The following is the reverse direction of the above Lemma 3.40.

Lemma 3.41. *For every propagation sequence $(\ell_0, x - y \leq 0) \to (\ell_1, x - y \leq c_1) \to \cdots \to (\ell_n, x - y \leq c_n)$ in the updatable timed automaton $\mathcal{A}_\mathcal{B}$, there is a run $(\ell_0, 0) \to (\ell_1, c_1) \to \cdots \to (\ell_n, c_n)$ in the bounded one-counter automaton $\mathcal{B}$.*

Proof. The proof is by induction on n. The base case $n = 0$ is trivial. Let $n > 0$ and suppose the lemma is true up to $n - 1$. Consider the propagation $(\ell_{n-1}, x - y \leq c_{n-1}) \to (\ell_n, x - y \leq c_n)$. This implies that, there is a transition $(\ell_n, g, up, \ell_{n-1})$ in $\mathcal{A}_\mathcal{B}$ with $up_x = x - p$ where $p = c_n - c_{n-1}$ and the constraint $\mathsf{pre}(x - y \leq c_{n-1}, g, up) = x - y \leq c_n$. By construction, every g is $x \leq b \wedge y \leq 0$. As $\mathsf{pre}(x - y \leq c_{n-1}, g, up)$ is non-trivial, firstly $c_n \leq b$ (otherwise Case 3 of Table 3.1 will apply) and secondly $0 \leq c_n$. To see this, suppose $c_n < 0$, then the constraint $x - y \leq c_n$ gets rewritten as $-c_n \leq y - x$ and the propagation would give $(\ell_n, -c_n \leq y - x)$ contrary to what was assumed. Now, consider the counter automaton $\mathcal{B}$. By induction hypothesis, there is a run up to (ℓ_{n-1}, c_{n-1}). From the construction, since there is the transition $(\ell_n, g, up, \ell_{n-1})$ in $\mathcal{A}_\mathcal{B}$, there is a transition (ℓ_{n-1}, p, ℓ_n) in $\mathcal{B}$. Recall that the constant c_n satisfies $0 \leq c_n \leq b$. Hence, there is a step $(\ell_{n-1}, c_{n-1}) \to (\ell_n, c_n)$ in $\mathcal{B}$, giving an extension to the run. □

It now remains to notice that the only transition that can generate infinitely many constraints during the propagation is the loop $\ell'_t \to \ell'_t$, since the other transitions between states coming from the counter automaton have a guard to cut out infinite propagations. For this to happen, some constraint needs to reach ℓ_t, and then propagate to ℓ'_t via the transition $\ell'_t \to \ell_t$.

Proposition 3.42. *The final state (ℓ_t) is reachable in the counter automaton $\mathcal{B}$ iff the smallest reduced $\mathcal{A}$-map of the updatable timed automaton $\mathcal{A}_\mathcal{B}$ contains a component that is infinite.*

Proof. Let $\mathcal{G}$ be the smallest reduced $\mathcal{A}_{\mathcal{B}}$-map.

Suppose the final state ℓ_t is reachable in $\mathcal{B}$, with a run $(\ell_0, c_0) \to \cdots \to (\ell_t, c_t)$. From Lemma 3.40, there is a propagation in $\mathcal{A}_{\mathcal{B}}$ adding $x - y \leq c_t$ to $\mathcal{G}(\ell_t)$. Due to the extra transitions $\ell'_t \to \ell_t$ and $\ell'_t \xrightarrow{x:=x,y:=y+1} \ell'_t$ in $\mathcal{A}_{\mathcal{B}}$ (with no guards), the constraint $x - y \leq c_t$ gets added to $\mathcal{G}(\ell'_t)$ and then for every integer $i \geq 0$, the constraint $x - y \leq c_t + i$ gets added to the set $\mathcal{G}(\ell'_t)$. Therefore, $\mathcal{G}$ becomes infinite.

Conversely, suppose ℓ be a state of the updatable timed automaton such that $\mathcal{G}(\ell)$ is infinite. Firstly $\ell \neq \ell'_0$, since there are no outgoing transitions from ℓ'_0 and therefore $\mathcal{G}(\ell'_0) = \emptyset$. Pick some state $\ell \in L \setminus \{\ell'_0, \ell'_t\}$. Every outgoing transition from ℓ has guard $x \leq b \wedge y \leq 0$, except in the case $\ell_0 \xrightarrow{x-y\leq 0} \ell'_0$. But since $\mathcal{G}(\ell'_0) = \emptyset$, this transition can be forgotten as far as propagation is concerned. Hence Table 3.1 ensures that, $d \leq b$ for every atomic constraint $x - y \leq d$ or $d \leq x - y$ propagated to $\mathcal{G}(\ell)$. Moreover, if at all there is a propagated constraint $d \leq y - x$ or $y - x \leq d$, then $d = 0$ since $y \leq 0$ is present in all outgoing guards. This shows that the number of diagonal constraints is finite in $\mathcal{G}(\ell)$. In the construction, a diagonal constraint $x - y \leq c$ always propagates as a diagonal since all updates are of the form $x := x - p$ and $y := y$. Coming to non-diagonals, since there are $x \leq b$ and $y \leq 0$ in all guards of outgoing transitions from ℓ, Case 1 of Table 3.1 disallows propagation of any other upper constraint to ℓ, and Case 2 of Table 3.1 bounds the possible constants of lower constraints $d \leq x$ or $d \leq y$ in $\mathcal{G}(\ell)$. This gives finite $\mathcal{G}(\ell)$ for $\ell \in L \cup \{\ell'_0\}$. Therefore, the only possibility is to have $\mathcal{G}(\ell'_t)$ infinite, due to the self-loop on ℓ'_t with update up being $x := x$ and $y := y + 1$. The infinite number of constraints arises due to $y := y + 1$ and hence should come from a constraint that involves y. Since there are no guards in this self-loop, $\mathsf{pre}(\varphi, \top, up) = up^{-1}(\varphi)$. For every constraint $\overline{\varphi}[c]$ involving y and the update up present in the self-loop, $up^{-1}(\overline{\varphi}[c]) = \overline{\varphi}[c - 1]$ if y occurs with a positive sign in φ and $\overline{\varphi}[c + 1]$ if y occurs with a negative sign in φ. There are three ways a constraint involving y reaches ℓ'_t during the propagation. One of them is $y \leq 0$ which could come from ℓ_t. But the pre of this is $y \leq -1$ and hence is trivial. Another possibility is from $\mathsf{pre}(0 \leq y, \top, up)$ used in the initialization step $\mathcal{G}^0$. But this gives $-1 \leq y$ which is again trivial. The only other way to have a propagation is to start from $x - y \leq 0 \in \mathcal{G}(\ell_0)$, reach some $x - y \leq c_t \in \mathcal{G}(\ell_t)$ with $0 \leq c_t \leq b$. This then passes on to $\mathcal{G}(\ell'_t)$. Starting from this, it then follows that the constraints $x - y \leq c_t + i$ for $i \geq 0$ all get added to $\mathcal{G}(\ell'_t)$. This gives a propagation sequence $(\ell_0, x - y \leq 0) \to \cdots \to (\ell_t, x - y \leq c_t)$. From Lemma 3.41, there is a corresponding run in the counter automaton $\mathcal{B}$, proving ℓ_t is reachable. □

Theorem 3.43. *Deciding termination of the reduced $\mathcal{A}$-map computation for a given updatable timed automaton $\mathcal{A}$ is in* PTIME *if the constants in $\mathcal{A}$ are encoded in unary, and* PSPACE*-complete if the constants are encoded in binary.*

Proof. The algorithm to detect termination given in Page 53 discusses the upper bound: PTIME when constants are in unary and PSPACE when the constants are in binary. Proposition 3.42 establishes the PSPACE lower-bound. Hence, the problem of deciding whether Algorithm 4 terminates, given an updatable timed automaton as input, is PSPACE-complete. □

3.5 Discussion

Given an updatable timed automaton $\mathcal{A}$, the goal of this chapter was to design a simulation relation for $\mathcal{A}$. Section 3.1 introduced the relation $\sqsubseteq_G$, first relating two valuations and then extending it to a relation between two configurations. The first condition for relating two configurations was that the states present in both of the configurations are the same. Therefore, the relation over configurations essentially reduces to the relation between the valuations present in the configurations. The relation $\sqsubseteq_G$ is parameterized by a set of atomic constraints G. When defining the relation between two configurations, instead of using a single G for every state, a state-specific set of atomic constraints $\mathcal{G}(q)$ is used. The challenge was to construct these sets $\mathcal{G}(q)$, for every state q of the input automaton $\mathcal{A}$, so that when plugged into the relation $\sqsubseteq_G$, the resulting relation $\sqsubseteq_{\mathcal{G}}$ becomes a simulation relation for $\mathcal{A}$.

Section 3.3 provided the appropriate constructions for these sets $\mathcal{G}(q)$ that make $\sqsubseteq_{\mathcal{G}}$ a simulation relation for Updatable Timed Automata. However, this parameter computation is not guaranteed to terminate for every updatable timed automaton. Since the relation $\sqsubseteq_{\mathcal{G}}$ can be used in a reachability algorithm only after the parameter is computed, it is important to know when this computation terminates and when it does not. Section 3.4 provided an algorithm for checking this.

From a theoretical point of view, Section 3.4 also showed that checking if this parameter computation terminates is, in fact, a PSPACE-complete problem. However, note that, the algorithm for deciding termination is not a separate procedure. This termination check is performed while computing the parameter itself. It only adds a comparison (Line 11 of Algorithm 4) between two constants, each time before deciding if a constraint should be added to a set $\mathcal{G}(q)$ or not, for some q. Therefore, this termination check does not result in an overhead for the overall algorithm.

For the restricted class of diagonal-free Timed Automata, there exists the well studied LU simulation relation. Since the relation $\sqsubseteq_G$ can also be defined for this class of automata, Section 3.2 compared these two relations. It was shown that the relation $\sqsubseteq_G$ can relate more valuations, and therefore more zones, than the LU simulation. This is due to the fact that the parameters for LU simulation do not consider the inequalities present in the guards and only consider the constants present. Whereas, since the parameter for $\sqsubseteq_G$ contains atomic constraints, it can also take the inequalities into account. However, since this improvement is not drastic, it may not result in large improvements in practice.

The improved parameter computation described in Section 3.3.2 shows that it might be possible to find even better parameters, that can result in coarser simulations. This chapter does not answer what is the 'best' possible choice for the sets $\mathcal{G}(q)$. This remains a problem to be looked at.

Finally, to get a reachability algorithm for a class of Timed Automata, being able to define a simulation relation is the first step. The next and final step is devising a procedure to check this relation between nodes of the zone graph. Since this check is performed everytime a new node is added to the zone graph, the simulation relation requires to be "efficiently checkable". This is the topic of the next chapter.

Chapter 4

Algorithm for checking simulation

Given a timed automaton as input, the reachability algorithm builds the *zone graph* corresponding to the input automaton. Naïve construction of this graph does not necessarily terminate. To ensure termination, the algorithm uses simulation relations. The previous chapter proposed a simulation relation for Updatable Timed Automata. However, only having a simulation relation available for a class of automata is not enough for using the reachability algorithm (Algorithm 1) for that class of automata. Another important step remains.

The nodes of the zone graph are pairs consisting of a state of the input automaton and a zone. A node n_1 is simulated by another node n_2 only if the states present in both the nodes are the same and the zone present in n_1 is simulated by the zone present in n_2. A state q of the input automaton is said to be reachable from a node n of the zone graph, if a node containing q is reachable from the node n. The idea to keep the zone graph finite is to not add 'unnecessary' nodes. That is, add a node to the zone graph only if some state is reachable from this new node that is not reachable from any other nodes (having the same state) present in the zone graph. This 'necessity' is precisely captured by simulation relations. A new node is not necessary to be added to the zone graph if it is *simulated by* another node present in the graph. This chapter describes how to check if this simulation holds or not.

The reason behind using a simulation relation is to ensure finiteness of the zone graph. But, not every simulation relation ensures finiteness. When simulation relations are 'finite', they guarantee finiteness of the zone graph built using that relation. Finiteness of a simulation relation means the following – every infinite sequence of zones contains two zones such that one zone is simulated by the preceding zone. This property therefore implies that when a *finite* simulation relation is used, the zone graph produced by the reachability algorithm is also finite. Section 4.1 proves that the relation $\sqsubseteq_G$ is finite when G is finite.

Whenever a new node gets discovered by the reachability algorithm, before adding it to the zone graph, it is checked first if it is simulated by any of the nodes already present in the graph. Therefore, the simulation relation is used frequently by the overall reachability algorithm. If this relation cannot be checked efficiently,

then clearly it will impact the overall performance of the reachability algorithm. An efficient algorithm for checking the simulation relation is therefore desired.

One reason behind the popularity of the LU simulation relation ($\sqsubseteq_{LU}$), defined by Behrmann et al. in [BBLP06] for diagonal-free Timed Automata, is the efficiency of checking this relation. The tools for checking reachability in diagonal-free Timed Automata including UPPAAL [LPY97] and TChecker [HP] use this LU simulation relation. Herbreteau et al. showed in [HSW16] that whether a zone is simulated by another with respect to $\sqsubseteq_{LU}$, can be checked in quadratic time. Chapter 3 proved that the relation $\sqsubseteq_G$ becomes a simulation relation (with appropriate G) for diagonal-free Timed Automata. Moreover, it also showed that the relation $\sqsubseteq_G$ relates more (valuations and hence more) zones than $\sqsubseteq_{LU}$. A question therefore may arise at this point: does this improvement make checking the relation $\sqsubseteq_G$ more difficult? Section 4.2 provides an algorithm for checking this relation in the restricted class of diagonal-free Timed Automata. It turns out, if the relation $\sqsubseteq_G$ does not hold between two zones, then it is due to at most two (non-diagonal) constraints present in G. Therefore, $\sqsubseteq_G$ can also be checked in quadratic time in the diagonal-free case, matching the theoretical complexity of checking $\sqsubseteq_{LU}$.

Finally, the main aim of devising an algorithm for checking the relation $\sqsubseteq_G$, when the input automaton contains diagonal constraints, is considered in Section 4.3. The algorithm to be proposed in this section, makes use of the algorithm devised for the diagonal-free case. It will be shown that if the set G contains d many diagonal constraints, then whether a zone is related to another zone with respect to $\sqsubseteq_G$, can be decided by checking $\mathcal{O}(2^d)$-many $\sqsubseteq_{G^{nd}}$ relations between different zones, where the set G^{nd} consists of all the non-diagonal constraints present in G. The existing algorithms for handling diagonal constraints all contain a step causing *mandatory* exponential blowup. Although checking $\sqsubseteq_G$ relation still comes with an exponential cost, this cost occurs in the worst case and in other situations it might be possible to conclude if this relation holds or not rather quickly. Is this exponential cost inevitable in the worst case? The final Section 4.4 talks about this. This section proves that deciding if $\sqsubseteq_G$ *does not hold* between two zones is NP-complete. This is proved by showing a polynomial time reduction from the 3-SAT problem.

The algorithms and results provided in this chapter are for the relation $\sqsubseteq_G$ where G is an arbitrary finite set of atomic constraints. Therefore, the contents of this chapter also hold when G is a specific set of atomic constraints, in particular, when G is part of a reduced $\mathcal{A}$-map. On the other hand, whether the underlying input automaton contains updates or not is not relevant for this chapter. Having updates only modifies how the successor nodes are computed while building the zone graph and does not alter when the relation $\sqsubseteq_G$ holds and when it does not.

4.1 The relation $\sqsubseteq_G$ is finite

A relation $\sqsubseteq$ is said to be *finite* if for every sequence of zones $\{Z_1, Z_2, Z_3, \dots\}$ there exist two zones Z_i and Z_j with $i > j$, such that, $Z_i \sqsubseteq Z_j$. The finiteness of a simulation relation guarantees the termination of the reachability algorithm employing that relation. As a first step towards making the relation $\sqsubseteq_G$ (introduced

and discussed in Chapter 3) suitable for use in a reachability algorithm, this section proves that the relation $\sqsubseteq_G$ is finite, when G is a finite set of atomic constraints.

Theorem 4.1. *Given a finite set of atomic constraints G, the relation $\sqsubseteq_G$ is finite.*

Given a zone Z, define the set $\downarrow_G Z$ to be $\{v \mid \exists v' \in Z.\ v \sqsubseteq_G v'\}$. For two zones Z_1, Z_2, if $Z_1 \not\sqsubseteq_G Z_2$ then $\downarrow_G Z_1 \neq \downarrow_G Z_2$. Similarly, given a valuation v, $\uparrow^G v$ denotes the set $\{v' \mid v \sqsubseteq_G v'\}$. For notational convenience, henceforth, whenever the underlying set G will be clear from context, only $\downarrow Z$ and $\uparrow v$ will be used.

Theorem 4.1 will be proved following the schema presented below:

Define an equivalence relation over the set of all valuations
(Definition 4.2)
↓
Number of equivalence classes (of that relation) is finite
(Lemma 4.3)
↓
Each $\downarrow Z$ is a union of those equivalence classes
(Lemma 4.7)
↓
There are only finitely many different $\downarrow Z$
↓
Theorem 4.1

The following is a relation between two valuations, given a constant $M \in \mathbb{N}$. The idea is to relate the "equivalent" valuations with respect to satisfaction of constraints whose constants are smaller than M.

Definition 4.2 (the relation $\simeq_M$)**.** *Given two valuations v, v', $v \simeq_M v'$ holds if the following conditions hold for every integer $0 \leq c \leq M$ and $0 \leq c' < M$:*

1. *for every pair of (distinct) clocks x, y:*
   (a) $v(x) - v(y) = c$ *iff* $v'(x) - v'(y) = c$,
   (b) $c' < v(x) - v(y) < c' + 1$ *iff* $c' < v'(x) - v'(y) < c' + 1$,
   (c) $v(x) - v(y) > M$ *iff* $v'(x) - v'(y) > M$,
2. *for every clock x:*
   (a) $v(x) = c$ *iff* $v'(x) = c$,
   (b) $c' < v(x) < c' + 1$ *iff* $c' < v'(x) < c' + 1$,
   (c) $v(x) > M$ *iff* $v'(x) > M$.

Given a finite set of atomic constraints G, let M_G be the maximum constant among all the constants present in the constraints belonging to G. To keep the notation simple, let $\simeq_G$ denote relation in Definition 4.2 with $M = M_G$.

Lemma 4.3. *Given a finite set of atomic constraints G, the relation $\simeq_G$ is an equivalence relation of finite index.*

Proof. Definition 4.2 implies that the relation $\simeq_G$ is an equivalence relation.

Given a finite set of atomic constraints G, let M_G denote the maximum constant present in the constraints of G. Consider the set $\Phi := \Phi_{xy} \cup \Phi_x$, where $\Phi_{xy} = \{x-y = c,\ c' < x-y < c'+1,\ x-y > M_G \mid x, y \text{ are non-zero clocks},\ c, c' \in \mathbb{N},\ c \leq M_G,\ c' < M_G\}$ and $\Phi_x = \{x = c,\ c' < x < c'+1,\ x > M_G \mid x \text{ is a non-zero clock},\ c, c' \in \mathbb{N},\ c \leq M_G,\ c' < M_G\}$. From Definition 4.2 it can be seen that, each equivalence class of $\simeq_G$ is defined by picking a constraint from Φ_{xy} for every pair of distinct non-zero clocks x, y and a constraint from Φ_x for every non-zero clock x. Since there are only finitely many clocks and Φ_{xy} and Φ_x both are finite, so is the number of equivalence classes of $\simeq_G$. □

The relation $\simeq_G$ therefore partitions the space of all valuations. Each equivalence class of the relation $\simeq_G$ will be called a G-region. The aim now is to show that for every zone Z, the set $\downarrow Z$ is a union of G-regions (Lemma 4.7). This will be proved using a reformulation of the set $\downarrow Z$. Note, if a valuation $v \notin \downarrow Z$ for some zone Z, then for every valuation v' satisfying $v \sqsubseteq_G v'$, $v' \notin Z$. Then from the definitions of the sets $\downarrow Z$ and $\uparrow v$ it follows that $v \notin \downarrow Z$ if and only if $\uparrow v \cap Z = \emptyset$. It will be shown that if $v \simeq_G v'$ then for every zone Z, $\uparrow v \cap Z \neq \emptyset$ if and only if $\uparrow v' \cap Z \neq \emptyset$ (Lemma 4.6). This result will imply that if a G-region intersects $\downarrow Z$, for some zone Z, then the G-region is in fact a subset of $\downarrow Z$ proving that $\downarrow Z$ is a union of G-regions. Before proving Lemma 4.6 the distance graph (see Page 22) representation of the set $\uparrow v$ is presented below.

Due to Corollary 3.6 and Corollary 3.10, it will be assumed throughout this chapter that the set of atomic constraints G contains at most one upper and at most one lower bound *non-diagonal* constraint per clock. This restriction cannot be assumed for diagonal constraints however, G may contain multiple upper and lower bound constraints involving the difference $x - y$, for example.

The graph $\mathsf{G}_{\uparrow v}$ to be constructed below, is the distance graph representation of the set $\uparrow v$, given a valuation and a finite set of atomic constraints G. The construction of $\mathsf{G}_{\uparrow v}$ needs to ensure: whenever a valuation satisfies all the constraints of $\mathsf{G}_{\uparrow v}$, the valuation belongs to the set $\uparrow v$ and vice versa (Lemma 4.4).

Let all the upper-bound diagonals in G involving the difference $x - y$, that v satisfies, be the set $G^u_{xy} := \{x - y \lhd_1 c_1,\ x - y \lhd_2 c_2,\ \ldots,\ x - y \lhd_k c_k\}$. Let $(\lhd, c)$ be such that $(\lhd, c) = \min\{(\lhd_i, c_i) \mid x - y \lhd_i c_i \in G^u_{xy}\}$. Then, for every valuation $v' \models x - y \lhd c \iff v' \models \varphi$, for every $\varphi \in G^u_{xy}$. Therefore, every valuation in $\uparrow v$ needs to satisfy the constraint $x - y \lhd c$. This is encoded in the edge in Figure 4.1.

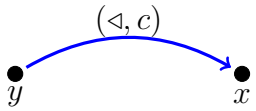


Figure 4.1: $\uparrow v$ contains the constraint $x - y \lhd c$

Let all the lower-bound diagonals in G involving the difference $x - y$, that v satisfies, be the set $G^\ell_{xy} := \{d_1 \lhd_1 x - y,\ d_2 \lhd_2 x - y,\ \ldots,\ d_j \lhd_j x - y\}$. Let $(\lhd, d)$ be such that $d = \max\{d_i \mid d_i \lhd_i x - y \in G^\ell_{xy}\}$ and $\lhd = <$ if $d < x - y \in G^\ell_{xy}$, otherwise

$\triangleleft = \leq$. Then again $v' \models d \triangleleft x - y \iff v' \models \varphi$ for every $\varphi \in G^{\ell}_{xy}$. The edge in Figure 4.2 encodes this constraint $d \triangleleft x - y$.

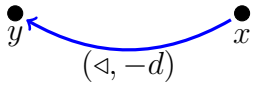


Figure 4.2: $\uparrow v$ contains the constraint $d \triangleleft x - y$

Now, consider the non-diagonal constraints present in G. Since G contains at most one upper bound non-diagonal constraint involving x, let G contains the constraint $\varphi := x \triangleleft c$. If the valuation $v \not\models \varphi$, then no constraint needs to be imposed on $\uparrow v$, hence no edges in the graph $\mathsf{G}_{\uparrow v}$. Whereas, if $v \models \varphi$, then whenever a valuation v' satisfies $v \sqsubseteq_G v'$, the inequality $v'(x) \leq v(x)$ must hold (Theorem 3.5). Also, since valuations always map clocks to non-negative values, $v'(x) \geq 0$ needs to hold as well. These two conditions are encoded in the pair of edges in Figure 4.3.

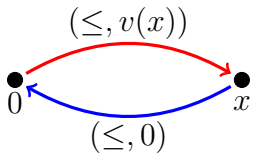


Figure 4.3: every $v' \in \uparrow v$ must satisfy $0 \leq v'(x) \leq v(x)$

Lastly, consider the lower bound non-diagonals present in G. If G contains no lower bound non-diagonal involving y, then no constraint needs to be imposed on $\uparrow v$. Otherwise, assume G contains the constraint $d \triangleleft y$. If $v \not\models d \triangleleft y$, then, in order for a valuation v' to satisfy $v \sqsubseteq_G v'$, v' needs to satisfy $v(y) \leq v'(y)$ (Lemma 3.8). This constraint $v(y) \leq v'(y)$ translates to the edge in Figure 4.4.

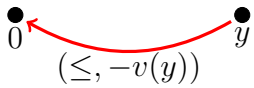


Figure 4.4: every $v' \in \uparrow v$ must satisfy $v(y) \leq v'(y)$

Otherwise, if the valuation $v \models d \triangleleft y$ then due to Lemma 3.7, whenever a valuation v' satisfies $v \sqsubseteq_G v'$, the valuation $v' \models d \triangleleft y$ as well. This condition translates to the edge depicted in Figure 4.5.

Note that, every blue edge ($\rightarrow$) in $\mathsf{G}_{\uparrow v}$ has integer weight. Whereas, every red edge ($\rightarrow$) has weight in terms of the valuation v, which may not be an integer.

The following Lemma proves that, the graph $\mathsf{G}_{\uparrow v}$ correctly represents the set $\uparrow v$. To recall the notation $[\![\mathsf{G}_{\uparrow v}]\!]$ (from Page 22) used in the following lemma, the set $[\![\mathsf{G}_{\uparrow v}]\!]$ consists of valuations that satisfy each of the constraints encoded in $\mathsf{G}_{\uparrow v}$.

Lemma 4.4. *Let v and v' be two valuations. Then, $v' \in [\![\mathsf{G}_{\uparrow v}]\!]$ iff $v \sqsubseteq_G v'$.*

Proof. ($\Rightarrow$) Let $v' \in [\![\mathsf{G}_{\uparrow v}]\!]$. Due to Proposition 3.3, in order to prove $v \sqsubseteq_G v'$, it is sufficient to show that $v \sqsubseteq_{\{\varphi\}} v'$ for every constraint $\varphi \in G$.

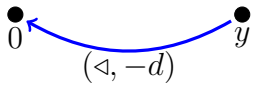


Figure 4.5: $\uparrow v$ contains the constraint $d \leq y$

Let $\varphi \in G$ be a diagonal constraint of the form $x - y \lhd_{xy} c_{xy}$. If $v \not\models \varphi$, then the relation $v \sqsubseteq_{\{\varphi\}} v'$ holds trivially. On the other hand, if $v \models \varphi$ then from the construction in Figure 4.1 it follows that $(\lhd, c) \leq (\lhd_{xy}, c_{xy})$, where $(\lhd, c)$ is the weight of the edge $y \to x$ in $\mathsf{G}_{\uparrow v}$. Then, since $v' \in [\![\mathsf{G}_{\uparrow v}]\!]$, the valuation $v' \models x - y \lhd c$ and hence $v' \models x - y \lhd_{xy} c_{xy}$. Similar argument holds if φ is of the form $d_{xy} \lhd_{xy} x - y$. Hence, in both of the cases $v \sqsubseteq_{\{\varphi\}} v'$.

Let $\varphi \in G$ be of the form $x \lhd_x c_x$. If $v \not\models \varphi$, then for every $\delta \in \mathbb{R}_{\geq 0}$ also $v+\delta \not\models \varphi$. Hence, the relation $v \sqsubseteq_{\{\varphi\}} v'$ holds trivially. Otherwise, if $v \models \varphi$ then the edges in Figure 4.3 are present in the graph $\mathsf{G}_{\uparrow v}$. Since the valuation v' satisfies these constraints, $0 \leq v'(x) \leq v(x)$ holds and therefore Theorem 3.5 implies $v \sqsubseteq_{\{\varphi\}} v'$.

Lastly, let $\varphi := d_y \lhd_y y \in G$. If $v \not\models \varphi$ then the graph $\mathsf{G}_{\uparrow v}$ contains the edge depicted in Figure 4.4. Since v' satisfies this constraint, $v'(y) \geq v(y)$. Therefore, from Theorem 3.9 it follows that $v \sqsubseteq_{\{\varphi\}} v'$. On the other hand, assume $v \models \varphi$. If there exist some other constraint $\varphi' := d'_y \lhd'_y y \in G$ such that $v \not\models \varphi'$ then the edge in Figure 4.4 still exist in the graph $\mathsf{G}_{\uparrow v}$ and therefore the result holds. Otherwise, if for every $\varphi := d_y \lhd_y y \in G$, $v \models \varphi$, then the graph $\mathsf{G}_{\uparrow v}$ contains the edge of Figure 4.5. Again, since v' satisfies this constraint, $v' \models d \lhd y$ and therefore $v' \models d_y \lhd_y y$, for every $d_y \lhd_y y \in G$. Hence, due to Theorem 3.9, $v \sqsubseteq_{\{\varphi\}} v'$ holds in this case as well.

($\Leftarrow$) Assuming $v \sqsubseteq_G v'$ it is required to show that $v' \in [\![\mathsf{G}_{\uparrow v}]\!]$, that is, v' satisfies all the constraints present in $\mathsf{G}_{\uparrow v}$. Since $v \sqsubseteq_G v'$, for every diagonal constraint φ, $v \models \varphi$ implies $v' \models \varphi$. Therefore, if the graph $\mathsf{G}_{\uparrow v}$ contains the constraints depicted in Figure 4.1 and Figure 4.2, then v' satisfies these. If G contains a constraint $x \lhd c$ such that $v \models x \lhd c$, then since $v \sqsubseteq_G v'$, from Theorem 3.5 it follows that $0 \leq v'(x) \leq v(x)$. Therefore, if $\mathsf{G}_{\uparrow v}$ contains the constraints depicted in Figure 4.3, then v' satisfies these. Lastly, if G contains a constraint of the form $d \lhd y$ such that $v \not\models d \lhd y$, then since $v \sqsubseteq_G v'$, it must hold that $v'(y) \geq v(y)$ (Lemma 3.8). Therefore, v' satisfies the constraint depicted in Figure 4.3. On the other hand, if for every $\varphi := d_y \lhd_y y \in G$, the valuation $v \models \varphi$ then it means $v \models d \lhd y$ as well. Then again, since $v \sqsubseteq_G v'$, it must hold that $v' \models d \lhd y$ (due to Lemma 3.7). Therefore, v' satisfies the constraint depicted in Figure 4.5. Hence, the valuation v' satisfies all the constraints of $\mathsf{G}_{\uparrow v}$ whenever $v \sqsubseteq_G v'$. □

An observation can be made at this point about the structure of two graphs $\mathsf{G}_{\uparrow v}$ and $\mathsf{G}_{\uparrow v'}$, when the valuations v, v' belong to the same G-region. The following result describes a similarity between these two graphs.

Lemma 4.5. *If v, v' are two valuations such that $v \simeq_G v'$, then every blue edge in $\mathsf{G}_{\uparrow v}$ has the same weight as the corresponding edge in the graph $\mathsf{G}_{\uparrow v'}$.*

Proof. Consider the edge depicted in Figure 4.1. Choose a diagonal constraint $\varphi := x - y \lhd c \in G$. If $v \models \varphi$, then due to the conditions 1a and 1b of Definition 4.2,

$v' \models \varphi$. Therefore, v' satisfies the same set of diagonal constraints that v does. Hence, $\mathsf{G}_{\uparrow v'}$ contains the edge of Figure 4.1 with the same weight as $\mathsf{G}_{\uparrow v}$. Similar argument holds for the type of edge depicted in Figure 4.2. The blue edge in Figure 4.3 is trivial, this will be present in $\mathsf{G}_{\uparrow v'}$ as well. Lastly, consider the edge depicted in Figure 4.5. The presence of this edge in $\mathsf{G}_{\uparrow v}$ means $v \models d_y \lhd_y y$ for every lower bound non-diagonal constraint $d_y \lhd_y y$ present in G. The conditions 2a and 2b of Definition 4.2 now imply that v' also satisfies all these constraints. Therefore, $\mathsf{G}_{\uparrow v'}$ contains the edge (of type Figure 4.5) with the same weight as in $\mathsf{G}_{\uparrow v}$. □

Given a zone Z, let G_Z be the distance graph representation of Z. Lemma 2.13 showed that the graph $\mathsf{min}(\mathsf{G}_{\uparrow v}, \mathsf{G}_Z)$ represents the set $[\![\mathsf{G}_{\uparrow v}]\!] \cap [\![\mathsf{G}_Z]\!]$, which is same as the set $\uparrow v \cap Z$ (since $\uparrow v = [\![\mathsf{G}_{\uparrow v}]\!]$, thanks to Lemma 4.4). Due to Lemma 2.12, the set $\uparrow v \cap Z$ is empty if the graph $\mathsf{min}(\mathsf{G}_{\uparrow v}, \mathsf{G}_Z)$ contains a negative cycle. The following Lemma proves that if $\mathsf{min}(\mathsf{G}_{\uparrow v}, \mathsf{G}_Z)$ contains a negative cycle, then for every valuation v' satisfying $v \simeq_G v'$, the graph $\mathsf{min}(\mathsf{G}_{\uparrow v'}, \mathsf{G}_Z)$ also contains a negative cycle. The intuition behind the proof is that every blue edge present in $\mathsf{G}_{\uparrow v}$ is also present in $\mathsf{G}_{\uparrow v'}$ (Lemma 4.5) and every red edge of $\mathsf{G}_{\uparrow v}$, when replaced by the corresponding edge from $\mathsf{G}_{\uparrow v'}$, keeps a negative cycle negative.

Lemma 4.6. *Let v, v' be two valuations such that $v \simeq_G v'$. Then, for every zone Z, if $\uparrow v \cap Z = \emptyset$ then $\uparrow v' \cap Z = \emptyset$.*

Proof. Let $\uparrow v \cap Z = \emptyset$. Due to Lemma 4.4, $[\![\mathsf{G}_{\uparrow v}]\!] = \uparrow v$. The graph $\mathsf{min}(\mathsf{G}_{\uparrow v}, \mathsf{G}_Z)$ contains a simple negative cycle, call it $\mathcal{C}$. Let gray ($\rightarrow$) edges in $\mathcal{C}$ be edges of G_Z and all red ($\rightarrow$) and blue ($\rightarrow$) edges be from $\mathsf{G}_{\uparrow v}$. The $\rightarrow$ edges contain integral weights, whereas, the $\rightarrow$ edges contain weights in terms of the valuation v, with possibly non-integer values. Note, every $\rightarrow$ edge in the graph $\mathsf{G}_{\uparrow v}$ (Figure 4.3 and Figure 4.4) is either outgoing or incoming to the vertex corresponding to the zero clock 0. Therefore, the cycle $\mathcal{C}$ can contain atmost two $\rightarrow$ edges.

If $\mathcal{C}$ contains no $\rightarrow$ edge then this Lemma follows from the previous Lemma 4.5. Suppose $\mathcal{C}$ contains a single $\rightarrow$ edge. If $\mathcal{C}$ contains the red edge in Figure 4.3 with weight $(\leq, v(x))$, then it means G contains a constraint $x \lhd c$ and the valuation $v \models x \lhd c$, therefore $v(x) \lhd c \leq M_G$. Since $v \simeq_G v'$, it follows from Definition 4.2 that $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$. This implies $v' \models x \lhd c$ as well, and hence $\mathsf{G}_{\uparrow v'}$ also contains this red edge. On the other hand, if $\mathcal{C}$ contains the red edge in Figure 4.4 with weight $(\leq, -v(y))$, then G must contain a constraint $d \lhd y$ and $v \not\models d \lhd y$. This also implies $v(y) \leq d \leq M_G$. Again, since $v \simeq_G v'$, Definition 4.2 implies that $\lfloor v(y) \rfloor = \lfloor v'(y) \rfloor$. Thus, $v' \not\models d \lhd y$ and $\mathsf{G}_{\uparrow v'}$ contains this red edge as well. Moreover, in either of the two cases, when the weight of the red edge in $\mathcal{C}$ is replaced by the weight of the corresponding edge in $\mathsf{G}_{\uparrow v'}$, the cycle $\mathcal{C}$ remains negative.

Now consider the case when $\mathcal{C}$ contains two $\rightarrow$ edges. In this case, the cycle $\mathcal{C}$ is of the form as depicted in Figure 4.6. The weight of the edge $x \leftarrow 0$ is $(\leq, v(x))$ (Figure 4.3) and the weight of the edge $0 \leftarrow y$ is $(\leq, -v(y))$ (Figure 4.4).

The claim is: this cycle $\mathcal{C}$ can be modified into a cycle $\mathcal{C}'$ that – (i) belongs to $\mathsf{min}(\mathsf{G}_{\uparrow v'}, \mathsf{G}_Z)$, and (ii) is also negative. This will then imply that $\uparrow v' \cap Z = \emptyset$. Since the weights of the $\rightarrow$ edges are same in the graph $\mathsf{G}_{\uparrow v'}$ as in $\mathsf{G}_{\uparrow v}$ (Lemma 4.5) these $\rightarrow$ as well as the $\rightarrow$ edges will be present in the graph $\mathsf{min}(\mathsf{G}_{\uparrow v'}, \mathsf{G}_Z)$ as well. Let $\mathcal{C}'$

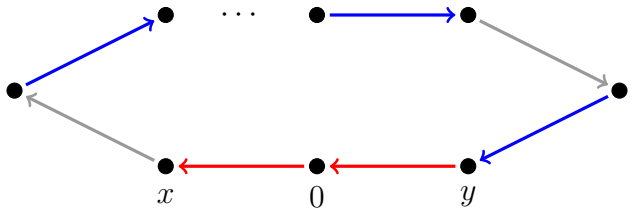


Figure 4.6: Negative cycle $\mathcal{C}$ present in $\uparrow v \cap Z$

be the cycle $\mathcal{C}$, with only the $\rightarrow$ edges replaced with the weights from $\mathsf{G}_{\uparrow v'}$ in place of the weights from $\mathsf{G}_{\uparrow v}$. These $\rightarrow$ edges are present in $\mathsf{min}(\mathsf{G}_{\uparrow v}, \mathsf{G}_Z)$ since $(\leq, v(x))$ and $(\leq, -v(y))$ are lesser than the weight of the corresponding edges in G_Z. Also, as argued in the single red edge case, the existence of these two red edges in the graph $\mathsf{G}_{\uparrow v}$ means $v(x) \leq M_G$ and $v(y) \leq M_G$, respectively. Since $\lfloor v(z) \rfloor = \lfloor v'(z) \rfloor$ for every clock z such that $v(z) \leq M_G$ and G_Z contains only integral weights, the pairs $(\leq, v'(x))$ and $(\leq, -v'(y))$ will still remain smaller than the weight of the corresponding edges in G_Z. Therefore, these *new* $\rightarrow$ edges, with the weights being $(\leq, v'(x))$ and $(\leq, -v'(y))$ respectively, belong to the graph $\mathsf{min}(\mathsf{G}_{\uparrow v'}, \mathsf{G}_Z)$ as well. The goal now is to show that the sum of the weights of edges of $\mathcal{C}'$ is also negative.

Let $\mathcal{S}_\mathcal{C}$ denote the sum of the edges of $\mathcal{C}$, $\mathcal{S}_{\mathcal{C}'}$ denote the sum of the edges of $\mathcal{C}'$ and $\mathcal{S}$ denote the sum of the non-red edges in $\mathcal{C}$. Note that the sum of the non-red edges in $\mathcal{C}'$ is also $\mathcal{S}$ (due to Lemma 4.5). Since every non-red edge has integral weight, the constant present in $\mathcal{S}$ is an integer. Assuming $\mathcal{S}_\mathcal{C} < 0$, the aim is to show $\mathcal{S}_{\mathcal{C}'} < 0$. Now, the sum $\mathcal{S}_\mathcal{C} = \mathcal{S} + (\leq, v(x)) + (\leq, -v(y)) = \mathcal{S} + (\leq, v(x) - v(y))$. Note that, since $v(x) \leq M_G$ and $v(y) \leq M_G$, the difference $v(x) - v(y) \leq M_G$ as well. Now, if $v(x) - v(y) \geq 0$ then Definition 4.2 implies that $\lfloor v(x) - v(y) \rfloor = \lfloor v'(x) - v'(y) \rfloor$. Otherwise, if $v(x) - v(y) < 0$ then $0 < v(y) - v(x) \leq M_G$ holds and therefore Definition 4.2 again implies $\lfloor v(x) - v(y) \rfloor = \lfloor v'(x) - v'(y) \rfloor$. Since the constant present in $\mathcal{S}$ is an integer and $\mathcal{S}_\mathcal{C} < 0$ it follows that $\mathcal{S}_{\mathcal{C}'} < 0$ as well.

Therefore, $\mathcal{C}'$ is also a negative cycle and hence $\uparrow v' \cap Z = \emptyset$. □

The above lemma implies that for every zone Z, the set $\downarrow Z$ is a union of G-regions (stated formally and proved in Lemma 4.7).

Lemma 4.7. *Given a zone Z, the set $\downarrow Z$ is a union of G-regions. In other words, if a G-region R is such that $R \cap \downarrow Z \neq \emptyset$ then $R \subseteq \downarrow Z$.*

Proof. Since $R \cap \downarrow Z \neq \emptyset$, let $v \in R \cap \downarrow Z$. Let v' be a valuation such that $v' \in R$ but $v' \notin \downarrow Z$. This means that $\uparrow v' \cap Z = \emptyset$. Lemma 4.6 then implies $\uparrow v \cap Z = \emptyset$ as well. This contradicts the assumption that $v \in \downarrow Z$. □

The above lemma, along with the fact that there are only finitely many distinct G-regions (Lemma 4.3), proves that the relation $\sqsubseteq_G$ is finite (Theorem 4.1).

Proof (of Theorem 4.1). Assume the relation $\sqsubseteq_G$ is not finite. Then, there exists an infinite sequence of zones $\mathcal{Z} := \{Z_1, Z_2, Z_3, \dots\}$ such that for every pair of integers $i > j$, $Z_i \not\sqsubseteq_G Z_j$. Now, $Z_i \not\sqsubseteq_G Z_j$ means there exists $v \in Z_i$ such that $v \notin \downarrow Z_j$. Therefore, $\downarrow Z_i \neq \downarrow Z_j$. Then, for every pair of zones Z_i, Z_j in $\mathcal{Z}$ with

$i > j$, $\downarrow Z_i \neq \downarrow Z_j$. This means there are infinitely many different $\downarrow Z$'s. But, the sets $\downarrow Z$ are unions of G-regions (Lemma 4.7) and there are only finitely many G-regions (Lemma 4.3). This is a contradiction. Therefore, $\sqsubseteq_G$ must be finite. □

Having established the finiteness of the relation $Z \sqsubseteq_G Z'$, the focus now turns to finding ways of checking when this relation actually holds. First up: the case when G contains only non-diagonal constraints. To emphasize this restriction on G, the next section will use the notation G^{nd} instead of G.

4.2 Checking $Z \sqsubseteq_{G^{nd}} Z'$ where G^{nd} is diagonal-free

This section considers the following problem: given two non-empty canonical zones Z, Z' and a finite set of *non-diagonal* constraints G^{nd}, how to check if $Z \sqsubseteq_{G^{nd}} Z'$? This section proposes Algorithm 5 for checking this relation through characterizing the negation of this relation. This algorithm will then be used in the next section to devise an algorithm for checking the relation $Z \sqsubseteq_G Z'$, when the set G will also be allowed to contain diagonal constraints.

The zones that will be considered throughout this section will be canonical, and therefore Proposition 2.11 will be referred to in various proofs. Given a canonical zone Z and two (possibly zero) distinct clocks x, y, the notation Z_{xy} (described on Page 21) will be used to denote the pair $(\lhd_{xy}, z_{xy})$, that is, the constraint $y - x \lhd_{xy} z_{xy}$ present in Z. Also, due to Corollary 3.6 and Corollary 3.10, it will be assumed that G^{nd} contains at most one lower and one upper bound constraint per clock.

The following theorem is the basis on which the algorithm for checking $Z \not\sqsubseteq_{G^{nd}} Z'$ will be built upon. This result implies that if $Z \not\sqsubseteq_{G^{nd}} Z'$, then the "non-relation" is due to at most two constraints of G^{nd}. That is, if $Z \not\sqsubseteq_{G^{nd}} Z'$ then either there is a constraint $\varphi \in G^{nd}$ for which $Z \not\sqsubseteq_{\{\varphi\}} Z'$ or there exist two constraints (one upper-bound and one lower-bound) φ_u, φ_ℓ in G^{nd} such that $Z \not\sqsubseteq_{\{\varphi_u, \varphi_\ell\}} Z'$.

Theorem 4.8. *Let Z, Z' be two non-empty canonical zones and $G^{nd} = G^u \cup G^\ell$ be a finite set of non-diagonal constraints with G^u containing only upper bound constraints and G^ℓ containing only lower bound constraints. Then, $Z \sqsubseteq_{G^{nd}} Z'$ iff the following two conditions hold:*

1. *$Z \sqsubseteq_{\{\varphi\}} Z'$ for every $\varphi \in G^{nd}$, or*
2. *$Z \sqsubseteq_{\{\varphi_u, \varphi_\ell\}} Z'$ for every $\varphi_u \in G^u$ and $\varphi_\ell \in G^\ell$.*

Proof. ($\Rightarrow$) If $Z \sqsubseteq_{G^{nd}} Z'$, then since $\{\varphi\} \subseteq G^{nd}$ for every φ and also $\{\varphi_u, \varphi_\ell\} \subseteq G^{nd}$ for every $\varphi_u \in G^u$, $\varphi_\ell \in G^\ell$, Proposition 3.2 implies $Z \sqsubseteq_{\{\varphi\}} Z'$ and $Z \sqsubseteq_{\{\varphi_u, \varphi_\ell\}} Z'$.

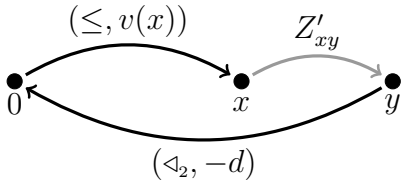
($\Leftarrow$) Assume $Z \sqsubseteq_{\{\varphi\}} Z'$ for every $\varphi \in G^{nd}$ and $Z \sqsubseteq_{\{\varphi_u, \varphi_\ell\}} Z'$ for every pair $\varphi_u \in G^u$ and $\varphi_\ell \in G^\ell$. The goal is to show $Z \sqsubseteq_{G^{nd}} Z'$. Assume $v \in Z$. It is sufficient to show that Z' contains a valuation v' such that $v \sqsubseteq_{G^{nd}} v'$. This will be proved by showing that the distance graph (see Page 22 for details about distance graphs) representing the zone $\uparrow v \cap Z'$ cannot contain a negative cycle.

To recall, given a finite set of atomic constraints (possibly containing diagonal constraints) G, $\uparrow^G v$ denotes the set of all valuations v' satisfying $v \sqsubseteq_G v'$. The distance graph representation of $\uparrow^G v$ has been presented in Lemma 4.4. Since this section deals with the set G^{nd} containing only non-diagonal constraints, for the rest of this section, $\uparrow v$ will denote the set of all valuations v' such that $v \sqsubseteq_{G^{nd}} v'$. The distance graph $\mathsf{G}_{\uparrow v}$ representing $\uparrow v$ will be a subgraph of the distance graph representing $\uparrow^G v$, described on Page 62. Only the edges described in Figure 4.3, Figure 4.4 and Figure 4.5 can be present in $\mathsf{G}_{\uparrow v}$. The edges described in Figure 4.1 and Figure 4.2 are based on diagonal constraints and therefore are not part of $\mathsf{G}_{\uparrow v}$.

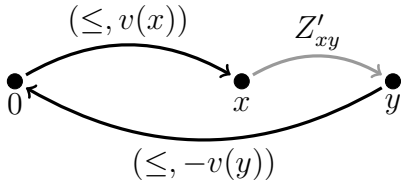
$\mathsf{G}_{\uparrow v}$ contains no other type of edge. The fact that the graph $\mathsf{G}_{\uparrow v}$ correctly represents $\uparrow v$ is stated in Lemma 4.4. The graph $\mathsf{min}(\mathsf{G}_{\uparrow v}, Z')$ represents the set $\uparrow v \cap Z'$ (Lemma 2.13). Note that, every edge present in $\mathsf{G}_{\uparrow v}$ is either incoming or outgoing from 0. Therefore, every (simple) cycle in $\mathsf{min}(\mathsf{G}_{\uparrow v}, Z')$ can contain atmost two edges coming from the graph $\mathsf{G}_{\uparrow v}$.

Moreover, since the zone Z' is canonical, in every negative cycle in $\mathsf{min}(\mathsf{G}_{\uparrow v}, Z')$, every sequence of edges $x_1 \xrightarrow{Z'_{x_1x_2}} x_2 \xrightarrow{Z'_{x_2x_3}} x_3 \to \cdots \to x_{n-1} \xrightarrow{Z'_{x_{n-1}x_n}} x_n$ can be replaced with the edge $x_1 \xrightarrow{Z'_{x_1x_n}} x_n$ and the cycle would remain negative, since the weight of this single edge is at most the sum of the weights of the edges present in the sequence. Therefore, it can be assumed that no two consecutive edges (in a negative cycle) come from Z'. Also, since $v \in \uparrow v$ and $\uparrow v \neq \emptyset$, no negative cycle in $\mathsf{min}(\mathsf{G}_{\uparrow v}, Z')$ can consist entirely of edges from $\mathsf{G}_{\uparrow v}$. Similarly, since Z' is assumed to be non-empty, no negative cycle can consist entirely of edges from Z' either.

These observations altogether imply that, if the graph $\mathsf{min}(\mathsf{G}_{\uparrow v}, Z')$ contains a negative cycle, it must be one of the cycles depicted in Figure 4.7a or 4.7b or 4.8 or 4.9a or 4.9b. In these cycles, the gray edges are from Z' and the remaining edges are from $\mathsf{G}_{\uparrow v}$. The weight of the edge $0 \to x$ is $(\leq, v(x))$ (according to Figure 4.3). The weight of the edge $y \to 0$ is either $(\lhd_2, -d)$ (according to Figure 4.5) or $(\leq, -v(y))$ (according to Figure 4.4). The weight of the gray edge in each of these cycles is the weight of the corresponding edge present in Z'.



(a) the valuation $v \models x \lhd_1 c$ and $v \models d \lhd_2 y$; the edge $y \to 0$ has weight $(\lhd_2, -d)$



(b) the valuation $v \models x \lhd_1 c$ but $v \not\models d \lhd_2 y$; the edge $y \to 0$ has weight $(\leq, -v(y))$

Figure 4.7: cycle in $\mathsf{min}(\mathsf{G}_{\uparrow v}, Z')$ containing two edges from $\mathsf{G}_{\uparrow v}$; the set G^{nd} contains two constraints $x \lhd_1 c$ and $d \lhd_2 y$

The claim now is that none of these cycles can be negative. Note that, the edges of the cycles in Figure 4.7a and Figure 4.7b get determined by only two constraints $x \lhd_1 c$ and $d \lhd_2 y$. To explain this observation, consider G' to be the set containing only these two constraints, that is $G' = \{x \lhd_1 c,\ d \lhd_2 y\}$. Also, let $\uparrow^{G'} v$

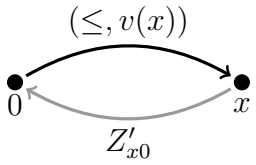
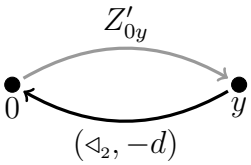
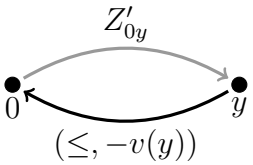


Figure 4.8: cycle in $\mathsf{min}(\mathsf{G}_{\uparrow v}, Z')$ with $0 \to x$ edge from $\mathsf{G}_{\uparrow v}$; the set G^{nd} contains the constraint $x \lhd_1 c$ and $v \models x \lhd_1 c$



(a) $y \to 0$ has weight $(\lhd_2, -d)$; the valuation $v \models d \lhd_2 y$



(b) $y \to 0$ has weight $(\leq, -v(y))$; the valuation $v \not\models d \lhd_2 y$

Figure 4.9: cycle in $\mathsf{min}(\mathsf{G}_{\uparrow v}, Z')$ with $y \to 0$ edge from $\mathsf{G}_{\uparrow v}$; G^{nd} contains the constraint $d \lhd_2 y$

denote the set of all valuations v' such that $v \sqsubseteq_{G'} v'$ and $\mathsf{G}'_{\uparrow v}$ be the distance graph representing $\uparrow^{G'} v$. Then, the two cycles in Figure 4.7 will also be a part of the graph $\mathsf{min}(\mathsf{G}'_{\uparrow v}, Z')$. This implies that neither of these two cycles can be negative, because otherwise $Z \not\sqsubseteq_{\{x \lhd c, d \lhd y\}} Z'$, which contradicts the assumption of the lemma.

Similarly, the cycle in Figure 4.8 gets determined by the constraint $x \lhd_1 c$. That is, if $G' = \{x \lhd_1 c\}$ and $\mathsf{G}'_{\uparrow v}$ is the distance graph representing $\uparrow^{G'} v$ then this cycle will also be present in the graph $\mathsf{min}(\mathsf{G}'_{\uparrow v}, Z')$. Therefore, this cycle can also not be negative, since for every $\varphi \in G$, the relation $Z \sqsubseteq_{\{\varphi\}} Z'$ holds. The same argument holds for the cycles in Figure 4.9a and 4.9b. These two cycles are due to the constraint $d \lhd_2 y$ and hence these cannot be negative as well.

Since these exhaust the set of all possible cycles in the graph $\mathsf{min}(\mathsf{G}_{\uparrow v}, Z')$ and none of these cycles can be negative, $\uparrow v \cap Z' \neq \emptyset$. Now, since v is an arbitrary valuation in Z, it follows that $Z \sqsubseteq_{G^{nd}} Z'$. This proves the theorem. □

Theorem 4.8 reduces the problem of checking $Z \sqsubseteq_{G^{nd}} Z'$ into two "smaller" checks: (i) $Z \sqsubseteq_{\{\varphi\}} Z'$ for every $\varphi \in G^{nd}$ or (ii) $Z \sqsubseteq_{\{\varphi_u, \varphi_\ell\}} Z'$ for every $(\varphi_u, \varphi_\ell) \in G^u \times G^\ell$. However, the problem remains: given Z and Z', how to check if these two "smaller" relations hold? The next three sections are devoted to this. Section 4.2.1 states the necessary and sufficient conditions that the two zones Z and Z' need to satisfy, so that $Z \not\sqsubseteq_{\{x \lhd_1 c\}} Z'$ holds. Similarly, Section 4.2.2 considers the lower bound constraint $d \lhd_2 y$ and characterizes $Z \not\sqsubseteq_{\{d \lhd_2 y\}} Z'$. Lastly, Section 4.2.3 deals with the two constraints' case. It states the conditions that imply (and are implied by) $Z \not\sqsubseteq_{\{x \lhd_1 c,\ d \lhd_2 y\}} Z'$. These three results are then put together in Section 4.2.4 to devise an overall algorithm for checking $Z \sqsubseteq_{G^{nd}} Z'$.

4.2.1 Checking $Z \not\preceq_{\{x \lhd_1 c\}} Z'$

The goal of this section is to characterize the non-relation $Z \not\preceq_{\{\varphi\}} Z'$, when φ is an upper bound constraint $x \lhd_1 c$. For the rest of this section, fix two non-empty canonical zones Z, Z' and the constraint $x \lhd_1 c$. The following proposition provides two "independent" conditions, that together imply $Z \not\preceq_{\{x \lhd_1 c\}} Z'$. By the word "independent" it is meant that the two conditions (present in the following proposition) require the existence of two possibly different valuations in Z.

Proposition 4.9. *$Z \not\preceq_{\{x \lhd_1 c\}} Z'$ iff the following two conditions hold:*

1. *$\exists v_1 \in Z$ such that $v_1 \models x \lhd_1 c$, and*
2. *$\exists v_2 \in Z$ such that $v_2(x) < v'_2(x)$, for all $v'_2 \in Z'$.*

Proof. ($\Rightarrow$) Assume $Z \not\preceq_{\{x \lhd_1 c\}} Z'$. From Definition 3.4 it follows that: $\exists v \in Z$ such that $\forall v' \in Z'$, $v \not\preceq_{\{x \lhd_1 c\}} v'$. Now, Theorem 3.5 implies that firstly $v \models x \lhd_1 c$ and also for every $v'_2 \in Z'$, the inequality $v(x) < v'_2(x)$ holds. Here the valuation v serves as both v_1 and v_2 mentioned in the proposition.

($\Leftarrow$) If $v_2 \models x \lhd_1 c$, then $v_2 \not\preceq_{\{x \lhd_1 c\}} v'_2$ for every $v'_2 \in Z'$ (due to Theorem 3.5). Hence $Z \not\preceq_{\{x \lhd_1 c\}} Z'$. Otherwise, $v_2 \not\models x \lhd_1 c$, that is $c \mathrel{\overline{\lhd_1}} v_2(x)$. From the assumption in this proposition: $v_1(x) \lhd_1 c$ and $v_2(x) < v'_2(x)$, for every $v'_2 \in Z'$. Coupled with $c \mathrel{\overline{\lhd_1}} v_2(x)$, this gives $v_1(x) < v'_2(x)$ for all $v'_2 \in Z'$. Hence, again from Theorem 3.5 it follows that $v_1 \not\preceq_{\{x \lhd_1 c\}} v'_2$, thereby giving $Z \not\preceq_{\{x \lhd_1 c\}} Z'$. □

Thanks to the previous proposition, the non-relation $Z \not\preceq_{\{x \lhd_1 c\}} Z'$ can now be ascertained by checking existence of two valuations satisfying the two conditions mentioned in the proposition. However, since zones contain infinitely many valuations it is still not clear how to check if these two conditions are indeed satisfied by Z and Z'. Two results will be presented next that formulate two conditions over Z, Z' that will be (necessary and) sufficient to conclude $Z \not\preceq_{\{x \lhd_1 c\}} Z'$.

Please recall the arithmetic defined (on Page 21) over the pairs $(\lhd, z)$ where $\lhd \in \{<, \leq\}$ and $z \in \mathbb{Z}$. This will be used in the proofs that follow.

The first condition mentioned in Proposition 4.9 requires the zone Z to contain a valuation v such that $v \models x \lhd_1 c$. To give an intuition about how to check the existence of such a valuation v, let Z be the zone $\{2 \leq x,\ 1 \leq y,\ y \leq 4,\ x - y \leq 3\}$ and the given constraint be $x \leq 3$. The question is: does Z contain a valuation v such that $v \models x \leq 3$? The answer to this question only depends on the constraint, describing the zone Z, that gives a lower-bound on the clock x. In this example, this constraint is $2 \leq x$. Since $[\![2 \leq x]\!] \cap [\![x \leq 3]\!] \neq \emptyset$ there indeed exists the valuation v, mapping $x \mapsto 2$ and $y \mapsto 3$, in the zone Z that also satisfies the constraint $x \leq 3$. This condition "$[\![2 \leq x]\!] \cap [\![x \leq 3]\!] \neq \emptyset$" can be rewritten as "$(\leq, 3) + (\leq, -2) \geq (\leq, 0)$". The first pair in the sum comes from the given constraint $x \leq 3$ and the second pair comes from the constraint $2 \leq x$ present in Z. Note that, if the given constraint is $x < 2$ instead, then Z does not contain any valuation satisfying this constraint, since for every valuation $v \in Z$, the value $v(x) \geq 2$. In this case, in terms of the sum of pairs, $(<, 2) + (\leq, -2) \not\geq (\leq, 0)$. The following lemma generalizes this condition for an arbitrary zone Z.

Lemma 4.10. *$\exists v \in Z$ such that $v \models x \lhd_1 c$ iff $(\lhd_1, c) + (\lhd_{x0}, z_{x0}) \geq (\leq, 0)$.*

Proof. ($\Rightarrow$) Assume there exists $v \in Z$ such that $v \models x \lhd_1 c$. This means the zone $Z \cap \{x \lhd_1 c\} \neq \emptyset$. Now, consider the distance graph representation G of the zone $Z \cap \{x \lhd_1 c\}$. Since the zone Z is non-empty, G cannot contain a negative cycle involving only edges from Z. Therefore, every possible negative cycle must contain the edge corresponding to the constraint $x \lhd_1 c$. Also, since Z is canonical, two consecutive edges from Z can be replaced by one. Hence, G contains a negative cycle if and only if there exists a negative cycle with only two edges with one of them being $0 \xrightarrow{(\lhd_1, c)} x$, that is, the cycle depicted in Figure 4.10.

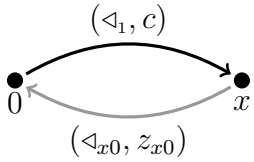


Figure 4.10: possible negative cycle in $Z \cap \{x \lhd_1 c\}$

Since the zone $Z \cap \{x \lhd_1 c\}$ is non-empty, the cycle depicted in Figure 4.10 cannot be negative, that is $(\lhd_1, c) + (\lhd_{x0}, z_{x0}) \geq (\leq, 0)$.

($\Rightarrow$) Conversely, if $(\lhd_1, c) + (\lhd_{x0}, z_{x0}) \geq (\leq, 0)$ then the cycle depicted in Figure 4.10 is not negative and therefore $Z \cap \{x \lhd_1 c\} \neq \emptyset$. Hence, there exists $v \in Z$ such that $v \models x \lhd_1 c$. □

A similar condition over the zones Z, Z' is required for the second condition mentioned in Proposition 4.9. This condition requires the zone Z to contain a valuation v such that for every valuation v' in Z', the inequality $v(x) < v'(x)$ holds. Note that Z will contain such a valuation if the lower bound for the clock x in Z is smaller than that of Z'. In that case, it will be possible to find a valuation in Z for which the value of x is strictly smaller than the lower bound (for x) allowed in Z'. For example, consider two zones $Z = \{2 < x, \dots\}$ and $Z' = \{3 \leq x, \dots\}$. Here, Z indeed contains a valuation v for which $v(x) = 2.5$ and for every $v' \in Z'$, the value $v(x) = 2.5 < 3 \leq v'(x)$. Also, in this case $(\leq, -3) < (<, -2)$, where the pair on the left side of the inequality comes from the constraint $3 \leq x$ (present in Z') and the pair on the right comes from the constraint $2 < x$ (present in Z). On the other hand, if $Z' = \{1 \leq x, \dots\}$, then Z' contains a valuation v' for which $v'(x) = 1$ and no valuation v belongs to Z for which $v(x) < v'(x)$, since $v(x) > 2$. Note that, in this case $(\leq, -1) \not< (<, -2)$. This condition is formalized in the following lemma.

Lemma 4.11. *$\exists v \in Z$ such that $\forall v' \in Z'$, $v(x) < v'(x)$ iff $(\lhd'_{x0}, z'_{x0}) < (\lhd_{x0}, z_{x0})$.*

Proof. ($\Leftarrow$) Assume $(\lhd'_{x0}, z'_{x0}) < (\lhd_{x0}, z_{x0})$. From this it follows that, either (i) $z'_{x0} < z_{x0}$ or (ii) $z'_{x0} = z_{x0}$, $\lhd'_{x0} = <$ and $\lhd_{x0} = \leq$. Case (i): $z'_{x0} < z_{x0} \implies -z_{x0} < -z'_{x0}$. Now, there exists $v \in Z$ such that $-z_{x0} < v(x) < -z'_{x0}$. Since for every $v' \in Z'$, $-z'_{x0} \lhd'_{x0} v'(x)$, it then follows that $v(x) < v'(x)$. Case (ii): since the zone Z is canonical, there exists $v \in Z$ such that $v(x) = -z_{x0}$ (Proposition 2.11). Then,

$v(x) = -z_{x0} = -z'_{x0} < v'(x)$, for every $v' \in Z'$. Therefore, in both of the cases, there exists $v \in Z$ such that for every $v' \in Z'$, $v(x) < v'(x)$.

($\Rightarrow$) To prove the contrapositive statement, let $(\lhd'_{x0}, z'_{x0}) \geq (\lhd_{x0}, z_{x0})$. This implies $z'_{x0} \geq z_{x0}$. If $\lhd'_{x0} = \leq$ then due to Proposition 2.11, Z' contains a valuation v' such that $-v'(x) = z'_{x0}$. This implies that for every $v \in Z$, the value $-v(x) \lhd_{x0} z_{x0} \leq z'_{x0} = -v'(x)$, that is, $v'(x) \leq v(x)$. On the other hand if $\lhd'_{x0} = <$ then $(\lhd'_{x0}, z'_{x0}) \geq (\lhd_{x0}, z_{x0})$ implies that either (i) $z'_{x0} > z_{x0}$ or (ii) $z'_{x0} = z_{x0}$ and $\lhd_{x0} = <$. In the first case, again due to Proposition 2.11, Z' contains a valuation v' such that $z'_{x0} > -v'(x) > z_{x0}$. This implies for every $v \in Z$, $-v(x) < -v'(x)$, that is, $v'(x) < v(x)$. In the second case, for every $v \in Z$ there will exist some valuation $v' \in Z'$ such that $-v(x) \leq -v'(x) < z'_{x0} = z_{x0}$ (thanks to Proposition 2.11). Therefore, there cannot exist $v \in Z$ such that $\forall v' \in Z'$, $v(x) < v'(x)$. □

Lemma 4.10 and Lemma 4.11 can now be combined with Proposition 4.9 to get a consolidated condition over the two zones Z, Z', such that, $Z \not\sqsubseteq_{\{x \lhd_1 c\}} Z'$.

Theorem 4.12. *$Z \not\sqsubseteq_{\{x \lhd_1 c\}} Z'$ iff the following two conditions hold:*

1. $(\lhd_1, c) + (\lhd_{x0}, z_{x0}) \geq (\leq, 0)$, *and*
2. $(\lhd'_{x0}, z'_{x0}) < (\lhd_{x0}, z_{x0})$.

The proof follows from Proposition 4.9, Lemma 4.10 and Lemma 4.11.

4.2.2 Checking $Z \not\sqsubseteq_{\{d \lhd_2 y\}} Z'$

The previous section provided two conditions over the zones Z, Z' that simultaneously need to hold in order to imply $Z \not\sqsubseteq_{\{x \lhd_1 c\}} Z'$. The aim of this section is to provide similar characterization for $Z \not\sqsubseteq_{\{d \lhd_2 y\}} Z'$. The following proposition describes such a formulation with the help of Theorem 3.9. For the rest of this section, fix two non-empty canonical zones Z, Z' and a constraint $d \lhd_2 y$.

Proposition 4.13. *$Z \not\sqsubseteq_{\{d \lhd_2 y\}} Z'$ iff the following two conditions hold:*

1. $\forall v' \in Z'$, $v' \not\models d \lhd_2 y$ *and*
2. $\exists v \in Z$ *such that* $\forall v' \in Z'$, $v'(y) < v(y)$.

Proof. ($\Rightarrow$) Assume $Z \not\sqsubseteq_{\{d \lhd_2 y\}} Z'$. Then Z contains a valuation v such that $v \notin {\downarrow}Z'$, that is, for every $v' \in Z'$, $v \not\sqsubseteq_{\{d \lhd_2 y\}} v'$. Then, for every $v' \in Z'$, from Theorem 3.9 it follows that: (i) $v' \not\models d \lhd_2 y$ and (ii) $v'(y) < v(y)$.

($\Leftarrow$) Let $v \in Z$ be the valuation such that for every $v' \in Z'$, $v'(y) < v(y)$. From Theorem 3.9, for a valuation $v' \in Z'$, the relation $v \sqsubseteq_{\{d \lhd_2 y\}} v'$ holds only if $v' \models d \lhd_2 y$. But the other hypothesis in the lemma does not permit this. Therefore, for every $v' \in Z'$, $v \not\sqsubseteq_{\{d \lhd_2 y\}} v'$ holds, proving $v \notin {\downarrow}Z'$ and hence $Z \not\sqsubseteq_{\{d \lhd_2 y\}} Z'$. □

Since the zones Z and Z' contain infinitely many valuations, it is again not clear how to check if the two conditions mentioned in Proposition 4.13 hold or not. The

next two results will provide two conditions that are, firstly, checkable given Z, Z' and that can replace the two corresponding conditions in the proposition above.

The first condition in the proposition requires that for every valuation v' in the zone Z', $v' \not\models d \lhd_2 y$. To give an example, consider a zone $Z' = \{y \leq 1, \dots\}$ and the constraint $2 \leq y$. The only relevant constraint in Z' is the constraint describing the upper bound for y. In this example, since for every $v' \in Z'$ the value $v'(y) \leq 1$, clearly, $2 \not\leq v'(y)$. The required condition holds if "a valuation with the maximum possible value for the clock y in Z', does not satisfy $d \lhd_2 y$". This translates to the formal condition: $(\leq, 1) + (\leq, -2) < (\leq, 0)$, where the first pair comes from the constraint $y \leq 1$ present in the zone Z' and the second pair comes from the given constraint $2 \leq y$. On the other hand, if the given constraint is $1 \leq y$ then Z' contains the valuation $v' : y \mapsto 1$ that satisfies the constraint $1 \leq y$. Note that, in this case, the sum of pairs $(\leq, 1) + (\leq, -1) \not< (\leq, 0)$.

Lemma 4.14. *$\forall v' \in Z'$, $v' \not\models d \lhd_2 y$ iff $(\lhd'_{0y}, z'_{0y}) + (\lhd_2, -d) < (\leq, 0)$.*

Proof. ($\Rightarrow$) For every $v' \in Z'$, $v' \not\models d \lhd_2 y$ means $Z' \cap \{d \lhd_2 y\} = \emptyset$. This implies the distance graph representing the zone $Z' \cap \{d \lhd_2 y\}$ contains a negative cycle. Following the similar arguments used in the proof of Lemma 4.10, it can be argued that the negative cycle must be as depicted in Figure 4.11. This cycle being negative implies the required inequality $(\lhd'_{0y}, z'_{0y}) + (\lhd_2, -d) < (\leq, 0)$.

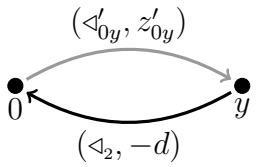


Figure 4.11: negative cycle in $Z' \cap \{d \lhd_2 y\}$

($\Leftarrow$) Assume $(\lhd'_{0y}, z'_{0y}) + (\lhd_2, -d) < (\leq, 0)$. This means the cycle in Figure 4.11 is negative and hence $Z' \cap \{d \lhd_2 y\} = \emptyset$. That is, for every $v' \in Z'$, $v' \not\models d \lhd_2 y$. □

The second condition in Proposition 4.13 requires Z to contain a valuation v satisfying $v'(y) < v(y)$ for every valuation $v' \in Z'$. This happens if the upper bound for the clock y in Z is bigger than the upper bound in Z'. No other constraints of the zones have any relevance for checking the required condition. For example, consider two zones $Z = \{y \leq 3, \dots\}$ and $Z' = \{y < 2, \dots\}$. The zone Z contains a valuation v for which $v(y) = 3$ and since for every $v' \in Z'$ the value $v'(y) < 2$, clearly $v'(y) < v(y)$. Note that, in this example $(<, 2) < (\leq, 3)$, where the first pair comes from the constraint $y < 2$ in Z' and the second one comes from $y \leq 3$ present in the zone Z. To consider a different example, let $Z' = \{y < 4\}$ instead. In this case, for every $v \in Z$, the value $v(y) \leq 3$. Since Z' contains a valuation v' with $v'(y) = 3.5$, Z contains no valuation v for which $v(y) > v'(y)$. Note that, $(<, 4) \not< (\leq, 3)$ in this case. This condition is generalized in the following lemma.

Lemma 4.15. *$\exists v \in Z$ such that $\forall v' \in Z'$, $v'(y) < v(y)$ iff $(\lhd'_{0y}, z'_{0y}) < (\lhd_{0y}, z_{0y})$.*

Proof. ($\Leftarrow$) Assume $(\lhd'_{0y}, z'_{0y}) < (\lhd_{0y}, z_{0y})$. There are two possibilities now: either (i) $z'_{0y} < z_{0y}$ or (ii) $z'_{0y} = z_{0y}$, $\lhd'_{0y} = <$ and $\lhd_{0y} = \leq$. Case (i): if $z'_{0y} < z_{0y}$ then there exists $v \in Z$ such that $z'_{0y} < v(y) < z_{0y}$. Now, since for every $v' \in Z'$, $v'(y) \lhd'_{0y} z'_{0y}$, it follows that $v'(y) < v(y)$. Case (ii): since Z' is canonical and $\lhd_{0y} = \leq$, there exists $v \in Z$ such that $v(y) = z_{0y}$. Then, since in this case $z'_{0y} = z_{0y}$, it follows that $v(y) = z'_{0y}$. Now, since $\lhd'_{0y} = <$, for every $v' \in Z'$, $v'(y) < z'_{0y}$. Therefore, it follows that $v'(y) < v(y)$ as well. Hence, in both of the cases there exists $v \in Z$ such that for every $v' \in Z'$, $v'(y) < v(y)$.

($\Rightarrow$) To prove the contrapositive, let $(\lhd_{0y}, z_{0y}) \leq (\lhd'_{0y}, z'_{0y})$. This implies the inequality $z_{0y} \leq z'_{0y}$. If $z_{0y} < z'_{0y}$, then due to Proposition 2.11, there exists $v' \in Z'$ such that for all $v \in Z$, $v(y) \leq z_{0y} < v'(y)$. On the other hand, assume $z_{0y} = z'_{0y}$. If $\lhd'_{0y} = \leq$, then again Z' contains v' such that for every $v \in Z$, $v(y) \leq z_{0y} = z'_{0y} = v'(y)$. Otherwise, if $\lhd'_{0y} = <$, then $\lhd_{0y} = <$ as well. Hence, Proposition 2.11 implies that for every $v \in Z$ there exists $v' \in Z'$ satisfying $v(y) \leq v'(y) < z'_{0y} = z_{0y}$. □

The following theorem combines Proposition 4.13 and subsequent Lemma 4.14 and Lemma 4.15 to derive the (checkable) conditions required for $Z \not\preceq_{\{d \lhd_2 y\}} Z'$.

Theorem 4.16. *$Z \not\preceq_{\{d \lhd_2 y\}} Z'$ iff the following two conditions hold:*

1. $(\lhd'_{0y}, z'_{0y}) + (\lhd_2, -d) < (\leq, 0)$ *and*
2. $(\lhd'_{0y}, z'_{0y}) < (\lhd_{0y}, z_{0y})$.

4.2.3 Checking $Z \not\preceq_{\{x \lhd_1 c,\ d \lhd_2 y\}} Z'$

Given two non-empty canonical zones Z, Z' and a set of non-diagonal constraints G^{nd}, Theorem 4.8 has reduced the problem of checking $Z \not\preceq_{G^{nd}} Z'$ to checking $Z \not\preceq_{\{\varphi\}} Z'$ for some $\varphi \in G^{nd}$ or $Z \not\preceq_{\{\varphi_u, \varphi_\ell\}} Z'$ for some pair of constraints φ_u, φ_ℓ, where φ_u is an upper bound constraint in G^{nd} and φ_ℓ is a lower bound constraint in G^{nd}. Sections 4.2.1 and 4.2.2 have given the conditions for checking $Z \not\preceq_{\{\varphi\}} Z'$. The aim of this section is to provide similar conditions for the remaining case of checking $Z \not\preceq_{\{\varphi_u, \varphi_\ell\}} Z'$, where φ_u is an upper bound and φ_ℓ is a lower bound constraint.

For the entirety of this section, fix two non-empty canonical zones Z, Z' and two constraints $x \lhd_1 c$ and $d \lhd_2 y$. Further, it is also assumed that $Z \sqsubseteq_{\{x \lhd_1 c\}} Z'$ and $Z \sqsubseteq_{\{d \lhd_2 y\}} Z'$. The next lemma is a first step towards characterizing $Z \not\preceq_{\{x \lhd_1 c,\ d \lhd_2 y\}} Z'$.

Lemma 4.17. *Assume $Z \sqsubseteq_{\{x \lhd_1 c\}} Z'$ and $Z \sqsubseteq_{\{d \lhd_2 y\}} Z'$. Then, $Z \not\preceq_{\{x \lhd_1 c,\ d \lhd_2 y\}} Z'$ iff $\exists v \in Z$ such that the following conditions hold:*

1. $v \models x \lhd_1 c$,
2. $(\leq, -v(y)) + (\leq, v(x)) + (\lhd'_{xy}, z'_{xy}) < (\leq, 0)$,
3. $(\lhd_2, -d) + (\leq, v(x)) + (\lhd'_{xy}, z'_{xy}) < (\leq, 0)$.

Proof. ($\Rightarrow$) Let $Z \not\preceq_{\{x \lhd_1 c,\ d \lhd_2 y\}} Z'$. Then, there exists $v \in Z$ such that $\mathsf{min}(\mathsf{G}_{\uparrow v}, Z')$ contains a negative cycle. This cycle must be of the form depicted in Figure 4.7, 4.8 or 4.9. But since $Z \sqsubseteq_{\{x \lhd_1 c\}} Z'$ and $Z \sqsubseteq_{\{d \lhd_2 y\}} Z'$, the cycle cannot be of the form

as in Figure 4.8 and 4.9. Therefore, the cycle must be one of the cycles depicted in Figure 4.7. Figure 4.12 depicts the general form of this cycle, call this cycle $\mathcal{C}$. The edges $0 \to x$ and $y \to 0$ are from $\mathsf{G}_{\uparrow v}$, and the edge $x \to y$ is from Z'.

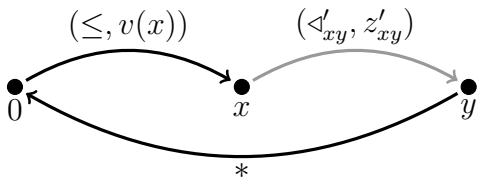


Figure 4.12: negative cycle $\mathcal{C}$ in the graph $\mathsf{min}(\mathsf{G}_{\uparrow v}, Z')$

The weights of the edges $0 \to x$ and $x \to y$ are $(\leq, v(x))$ (see Figure 4.3) and $(\lhd'_{xy}, z'_{xy})$ respectively. Whereas, the weight of the edge $y \to 0$ is either (i) $(\leq, -v(y))$ (see Figure 4.5) or (ii) $(\lhd_2, -d)$ (see Figure 4.4).

Since the graph $\mathsf{G}_{\uparrow v}$ contains the edge $0 \to x$ with weight $(\leq, v(x))$, it implies (from the construction presented in Figure 4.3) condition 1, that is, $v \models x \lhd_1 c$. The satisfaction of the remaining two conditions are explained below. $\mathcal{S}_{v(y)}$ is used below to denote the sum $(\leq, -v(y)) + (\leq, v(x)) + (\lhd'_{xy}, z'_{xy})$, and $\mathcal{S}_d$ is used to denote the sum $(\lhd_2, -d) + (\leq, v(x)) + (\lhd'_{xy}, z'_{xy})$.

Case (i). In this case, the weight of the edge $y \to 0$ is $(\leq, -v(y))$. Since the cycle $\mathcal{C}$ is negative, $\mathcal{S}_{v(y)} < (\leq, 0)$. Also, in this case $v \not\models d \lhd_2 y$, that is, $v(y) \overline{\lhd_2} d$. Then the claim is that $(\lhd_2, -d) < (\leq, -v(y))$. This is because, either $v(y) < d$, that is, $-d < -v(y)$ and thus the claim follows, or $v(y) = d$, in which case $\overline{\lhd_2}$ must be the weak inequality $\leq$ and hence $\lhd_2$ is the strict inequality $<$, therefore the claim follows again. Therefore, from the claim it follows that $\mathcal{S}_d < \mathcal{S}_{v(y)} < (\leq, 0)$.

Case (ii). In this case, the weight of the edge $y \to 0$ is $(\lhd_2, -d)$. Again, since the cycle $\mathcal{C}$ is negative, it follows that $\mathcal{S}_d < (\leq, 0)$. Now, in this case $v \models d \lhd_2 y$, hence, $-v(y) \lhd_2 -d$. Then, the claim is $(\leq, -v(y)) \leq (\lhd_2, -d)$ holds. This is because, either $-v(y) < -d$ and then the claim holds, or $-v(y) = -d$, in which case $\lhd_2$ is the weak inequality $\leq$ and hence the claim follows in this case as well. Therefore, from the claim it follows that $\mathcal{S}_{v(y)} \leq \mathcal{S}_d < (\leq, 0)$.

Therefore, in both of the cases, $\mathcal{S}_{v(y)} < (\leq, 0)$ as well as $\mathcal{S}_d < (\leq, 0)$.

($\Leftarrow$) If Z contains a valuation v satisfying the three conditions of the lemma, then the cycle in Figure 4.12 is negative, for both possible values of $*$. Therefore, $\mathsf{min}(\mathsf{G}_{\uparrow v}, Z')$ contains a negative cycle proving $Z \not\sqsubseteq_{\{x \lhd_1 c,\ d \lhd_2 y\}} Z'$. □

Unlike Proposition 4.9 and Proposition 4.13, the previous lemma does not provide "independent" conditions that are required for $Z \not\sqsubseteq_{\{x \lhd_1 c,\ d \lhd_2 y\}} Z'$ to hold. In the above lemma, the same valuation $v \in Z$ needs to satisfy all the three conditions. Given Z, Z' it is again not clear how to check whether Z contains such a valuation. The following proposition further characterizes the three conditions of Lemma 4.17 and states that the three conditions can actually be made independent of each other. That is, if there exist three different valuations satisfying each of the three conditions separately, then there will also exist a single valuation satisfying all the three conditions simultaneously.

Proposition 4.18. *Z contains a valuation v that satisfies the following properties:*

i. $v \models x \lhd_1 c$,

ii. $(\leq, -v(y)) + (\leq, v(x)) + (\lhd'_{xy}, z'_{xy}) < (\leq, 0)$,

iii. $(\lhd_2, -d) + (\leq, v(x)) + (\lhd'_{xy}, z'_{xy}) < (\leq, 0)$.

if and only if the following three conditions hold -

a. $\exists v_1 \in Z$ *such that* $v_1 \models x \lhd_1 c$,

b. $\exists v_2 \in Z$ *such that* $(\leq, -v_2(y)) + (\leq, v_2(x)) + (\lhd'_{xy}, z'_{xy}) < (\leq, 0)$,

c. $\exists v_3 \in Z$ *such that* $(\lhd_2, -d) + (\leq, v_3(x)) + (\lhd'_{xy}, z'_{xy}) < (\leq, 0)$.

The forward direction of Proposition 4.18 follows immediately. The valuation v that satisfies the three conditions $(i), (ii), (iii)$, can serve as the three valuations v_1, v_2, v_3 present in the conditions $(a), (b), (c)$ respectively. The reverse direction however requires more arguments. The following three results construct three zones, one zone corresponding to each of the conditions (a), (b) and (c), such that, each zone represents the set of *all* valuations that satisfy the corresponding condition. These three zones then help formulate the subsequent result (Lemma 4.22) which directly implies the backward direction of Proposition 4.18.

Lemma 4.19. *Let $\mathcal{Z}_1$ be the zone $\{x \lhd_1 c\}$. Then, $v_1 \in \mathcal{Z}_1$ iff $v_1 \models x \lhd_1 c$.*

The proof of this lemma follows directly from the definition of zones.

Lemma 4.20. *Let $\mathcal{Z}_2$ be the zone given by $\{x - y \lhd^{z_2} -z'_{xy}\}$, where $\lhd^{z_2} = \overline{\lhd'_{xy}}$. Then, $v_2 \in \mathcal{Z}_2$ iff $(\leq, -v_2(y)) + (\leq, v_2(x)) + (\lhd'_{xy}, z'_{xy}) < (\leq, 0)$.*

Proof. ($\Rightarrow$) Let $v_2 \in \mathcal{Z}_2$. Then, $v_2(x) - v_2(y) \lhd^{z_2} -z'_{xy}$. Therefore $v_2(x) - v_2(y) + z'_{xy} \leq 0$. When $v_2(x) - v_2(y) + z'_{xy} < 0$, the lemma follows. Otherwise, if $v_2(x) - v_2(y) + z'_{xy} = 0$, then $\lhd^{z_2}$ has to be $\leq$. This implies $\lhd'_{xy}$ is the strict inequality $<$, due to which the lemma follows for this case too.

($\Leftarrow$) Let v_2 be a valuation satisfying $(\leq, -v_2(y)) + (\leq, v_2(x)) + (\lhd'_{xy}, z'_{xy}) < (\leq, 0)$. It needs to be shown that $v_2 \in \mathcal{Z}_2$. Now, if $v_2(x) - v_2(y) < z'_{xy}$ then clearly, $v_2 \models x - y \lhd^{z_2} -z'_{xy}$ holds. On the other hand, if $v_2(x) - v_2(y) = z'_{xy}$, then it implies that $\lhd'_{xy} = <$ and hence $\lhd^{z_2} = \leq$. Therefore, again $v_2 \models x - y \lhd^{z_2} -z'_{xy}$ holds. □

Lemma 4.21. *Let $\mathcal{Z}_3$ be the zone given by $\{x \lhd^{z_3} d - z'_{xy}\}$, where $\lhd^{z_3}$ is $\leq$ if at least one of $\lhd_2$ or $\lhd'_{xy}$ is $<$, otherwise $\lhd^{z_3}$ is $<$. Then, a valuation $v_3 \in \mathcal{Z}_3$ iff v_3 satisfies the inequality $(\lhd_2, -d) + (\leq, v_3(x)) + (\lhd'_{xy}, z'_{xy}) < (\leq, 0)$.*

Proof. ($\Rightarrow$) Let $v_3 \in \mathcal{Z}_3$, then $v_3(x) \lhd^{z_3} d - z'_{xy}$. If $v_3(x) < d - z'_{xy}$, then the lemma follows. Otherwise, if $v_3(x) = d - z'_{xy}$, then it means $\lhd^{z_3} = \leq$. Which, in turn, means that $\lhd_2 = <$ or $\lhd'_{xy} = <$. Therefore, the result follows again.

($\Leftarrow$) Let v_3 be a valuation satisfying $(\lhd_2, -d) + (\leq, v_3(x)) + (\lhd'_{xy}, z'_{xy}) < (\leq, 0)$. Again, if $-d + v_3(x) + z'_{xy} < 0$ then $v_3 \models x \lhd^{z_3} d - z'_{xy}$, for both values of $\lhd^{z_3}$. On the other hand, if $-d + v_3(x) + z'_{xy} = 0$ then at least one of $\lhd_2$ or $\lhd'_{xy}$ is the strict inequality $<$. Therefore, $\lhd^{z_3}$ is $\leq$, and hence $v_3 \models x \lhd^{z_3} d - z'_{xy}$ as well. □

The following lemma is the final result required to prove the backward direction of Proposition 4.18. This result states that if Z contains no single valuation that satisfies the conditions $(i), (ii), (iii)$ in Proposition 4.18, then there must be at least one condition among $(a), (b), (c)$ (in Proposition 4.18) that is not satisfied by any valuation belonging to the zone Z.

Lemma 4.22. *$Z \cap \mathcal{Z}_1 \cap \mathcal{Z}_2 \cap \mathcal{Z}_3 = \emptyset$ iff $Z \cap \mathcal{Z}_i = \emptyset$ for some $i \in \{1, 2, 3\}$.*

Proof. ($\Leftarrow$) This direction is immediate.

($\Rightarrow$) To prove this direction of the lemma, it will be useful to consider the distance graph representations of the three zones $\mathcal{Z}_1$ (Figure 4.13), $\mathcal{Z}_2$ (Figure 4.14) and $\mathcal{Z}_3$ (Figure 4.15).

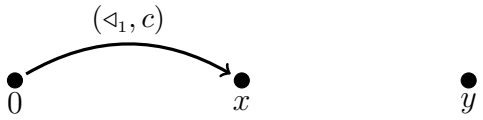


Figure 4.13: distance graph representation of $\mathcal{Z}_1$

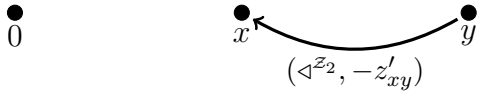


Figure 4.14: distance graph representation of $\mathcal{Z}_2$

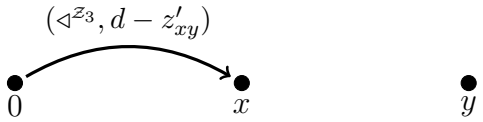


Figure 4.15: distance graph representation of $\mathcal{Z}_3$

Since $Z \cap \mathcal{Z}_1 \cap \mathcal{Z}_2 \cap \mathcal{Z}_3 = \emptyset$, the distance graph representing the zone $Z \cap \mathcal{Z}_1 \cap \mathcal{Z}_2 \cap \mathcal{Z}_3$ contains a negative cycle. Since the zone Z is non-empty, this cycle cannot consist only of edges from Z, it must contain at least one edge from $\mathcal{Z}_1, \mathcal{Z}_2$ or $\mathcal{Z}_3$. Furthermore, from the distance graphs of the zones $\mathcal{Z}_1, \mathcal{Z}_2, \mathcal{Z}_3$ it follows that no (simple) cycle can contain edges from both $\mathcal{Z}_i$ and $\mathcal{Z}_j$ where $i, j \in \{1, 2, 3\}$ and $i \neq j$. This means, every (simple) negative cycle in the distance graph representing the zone $Z \cap \mathcal{Z}_1 \cap \mathcal{Z}_2 \cap \mathcal{Z}_3$ must contain exactly one edge from either $\mathcal{Z}_1$ or $\mathcal{Z}_2$ or $\mathcal{Z}_3$. This means these negative cycles will also be present in the distance graphs representing $Z \cap \mathcal{Z}_1$ or $Z \cap \mathcal{Z}_2$ or $Z \cap \mathcal{Z}_3$. Therefore, if $Z \cap \mathcal{Z}_1 \cap \mathcal{Z}_2 \cap \mathcal{Z}_3$ is empty then so is the zone $Z \cap \mathcal{Z}_i$ for some $i \in \{1, 2, 3\}$. □

All required results are now in place to prove Proposition 4.18.

Proof (of Proposition 4.18). ($\Rightarrow$) Assume $v \in Z$ is the valuation that satisfies all the three conditions $(i), (ii)$ and (iii). Then, choosing $v_1 = v_2 = v_3 = v$ proves this direction, thanks to Lemma 4.19, Lemma 4.20 and Lemma 4.21.

($\Leftarrow$) Let $\mathcal{Z}_1, \mathcal{Z}_2$ and $\mathcal{Z}_3$ be the zones described in Lemma 4.19, Lemma 4.20 and Lemma 4.21 respectively. Then, $v_1 \in Z \cap \mathcal{Z}_1$ (due to Lemma 4.19), $v_2 \in Z \cap \mathcal{Z}_2$ (due to Lemma 4.20) and $v_3 \in Z \cap \mathcal{Z}_3$ (due to Lemma 4.21). From Lemma 4.22 it then follows that $Z \cap \mathcal{Z}_1 \cap \mathcal{Z}_2 \cap \mathcal{Z}_3 \neq \emptyset$. Therefore, there exists $v \in Z$ such that $v \in \mathcal{Z}_i$ for every $i \in \{1, 2, 3\}$. This proves the lemma. □

Lemma 4.17 gave three conditions that need to be satisfied by a single valuation $v \in Z$ in order to have $Z \not\sqsubseteq_{\{x \lhd_1 c,\ d \lhd_2 y\}} Z'$. Proposition 4.18 then proved that it is sufficient if three (possibly all different) valuations satisfy each of the three conditions. However, it is still not immediate how to check if such three valuations exist in Z. The next set of results provide the conditions over the zones Z, Z' that are checkable and will help answer the question of whether $Z \not\sqsubseteq_{\{x \lhd_1 c,\ d \lhd_2 y\}} Z'$.

Condition (a) of Proposition 4.18 is same as the condition (1) of Proposition 4.9. Lemma 4.10 provides the condition over Z that is (necessary and) sufficient for this condition to hold. The seemingly different condition (c) is also similar to (a) and the same lemma can be used to devise the required conditions. This similarity is due to the zone formulation in Lemma 4.21.

Lemma 4.23. *$\exists v \in Z$ such that $(\lhd_2, -d) + (\leq, v(x)) + (\lhd'_{xy}, z'_{xy}) < (\leq, 0)$ iff the inequality $(\lhd'_{xy}, z'_{xy}) + (\lhd_2, -d) < (\lhd_{x0}, z_{x0})$ holds.*

Proof. The valuation v satisfies $(\lhd_2, -d) + (\leq, v(x)) + (\lhd'_{xy}, z'_{xy}) < (\leq, 0)$ iff v also satisfies the constraint $x \lhd^{z_3} d - z'_{xy}$, where $\lhd^{z_3}$ is $\leq$ if at least one of $\lhd_2$ or $\lhd'_{xy}$ is $<$, otherwise $\lhd^{z_3}$ is $<$ (Lemma 4.21). Lemma 4.10 further implies that there exists $v \models x \lhd^{z_3} d - z'_{xy}$ iff $(\lhd_{x0}, z_{x0}) + (\lhd^{z_3}, d - z'_{xy}) \geq (\leq, 0)$. From the definition of the relation $\lhd^{z_3}$ it follows that $(\lhd^{z_3}, d - z'_{xy}) = (\overline{\lhd_2}, d) + (\overline{\lhd'_{xy}}, -z'_{xy})$. That is, the following inequality holds: $(\lhd_{x0}, z_{x0}) + (\overline{\lhd_2}, d) + (\overline{\lhd'_{xy}}, -z'_{xy}) \geq (\leq, 0)$. This then implies either of the following two conditions hold: (i) either $z_{x0} + d - z'_{xy} > 0$, that is, $z'_{xy} - d < z_{x0}$ or (ii) $z_{x0} + d - z'_{xy} = 0$ and $\lhd_{x0} = \overline{\lhd_2} = \overline{\lhd'_{xy}} = \leq$, these together imply $z'_{xy} - d = z_{x0}$ and $\lhd_{x0} = \leq, \lhd'_{xy} = \lhd_2 = <$. These two conditions together implies the inequality: $(\lhd'_{xy}, z'_{xy}) + (\lhd_2, -d) < (\lhd_{x0}, z_{x0})$. This proves the lemma. □

The next lemma characterizes the remaining condition (b) of Proposition 4.18. This condition requires Z to contain a valuation v such that the following inequality holds: $(\leq, -v(y)) + (\leq, v(x)) + (\lhd'_{xy}, z'_{xy}) < (\leq, 0)$. This inequality depends on the constraints putting an upper bound on the difference $y - x$ in the zones Z and Z'. The left hand side of the inequality can be simplified as $(\leq, -(v(y) - v(x))) + (\lhd'_{xy}, z'_{xy})$. To give an example, consider two canonical zones $Z = \{y - x \leq 3, \dots\}$ and $Z' = \{y - x < 2, \dots\}$. Here, because of canonicity, Z contains a valuation v such that $v(y) - v(x) = 3$ (due to Proposition 2.11). Since $(\lhd'_{xy}, z'_{xy}) = (<, 2)$, it follows that $(\leq, -3) + (<, 2) < (\leq, 0)$. The first pair on the left hand side comes from the constraint $y - x \leq 3$ present in Z and the second pair comes from $y - x < 2$ present in the zone Z'. Note that, in this example $(<, 2) < (\leq, 3)$. This condition is generalized in the next lemma. To give another example, let Z be same as the example before and $Z' = \{y - x < 4, \dots\}$ instead. Then, since for every valuation $v \in Z$, $v(y) - v(x) \leq 3$, the sum $(\leq, -(v(y) - v(x))) + (<, 4) \not< (\leq, 0)$. Note that, in this example, $(<, 4) \not< (\leq, 3)$.

Lemma 4.24. *$\exists v \in Z$ satisfying $(\leq, -v(y)) + (\leq, v(x)) + (\lhd'_{xy}, z'_{xy}) < (\leq, 0)$ iff $(\lhd'_{xy}, z'_{xy}) < (\lhd_{xy}, z_{xy})$.*

Proof. ($\Rightarrow$) Let $(\leq, -v(y)) + (\leq, v(x)) + (\lhd'_{xy}, z'_{xy}) < (\leq, 0)$. Then, there are two possibilities: (i) $z'_{xy} < v(y) - v(x)$ or (ii) $z'_{xy} = v(y) - v(x)$ and $\lhd'_{xy} = <$. When $z'_{xy} < v(y) - v(x)$, since $v(y) - v(x) \lhd_{xy} z_{xy}$, the result follows. Otherwise, when $z'_{xy} = v(y) - v(x)$, $\lhd'_{xy} = <$. Again, $v(y) - v(x) \lhd_{xy} z_{xy}$. If $\lhd_{xy} = \leq$ then the result follows. On the other hand, if $\lhd_{xy} = <$ then $z'_{xy} = v(y) - v(x) < z_{xy}$ and hence the result follows.

($\Leftarrow$) Let $(\lhd'_{xy}, z'_{xy}) < (\lhd_{xy}, z_{xy})$. If $z'_{xy} < z_{xy}$ then there must exist $v \in Z$ such that $z'_{xy} < v(y) - v(x) \lhd_{xy} z_{xy}$ (Proposition 2.11) and therefore the result follows. On the other hand, if $z'_{xy} = z_{xy}$ then it must be the case that $\lhd'_{xy} = <$ and $\lhd_{xy} = \leq$. Since $\lhd_{xy} = \leq$, Z will then contain a valuation v such that $v(y) - v(x) = z_{xy}$. The result then follows from the fact that $\lhd'_{xy} = <$. □

The results proved in this section, finally, lead to the following set of conditions that ensure $Z \not\sqsubseteq_{\{x \lhd_1 c,\ d \lhd_2 y\}} Z'$. The proof follows from Lemma 4.17, Proposition 4.18, Lemma 4.10, Lemma 4.24 and Lemma 4.23.

Theorem 4.25. *$Z \not\sqsubseteq_{\{x \lhd_1 c,\ d \lhd_2 y\}} Z'$ iff the following three conditions hold:*

1. $(\lhd_{x0}, z_{x0}) + (\lhd_1, c) \geq (\leq, 0)$,
2. $(\lhd'_{xy}, z'_{xy}) < (\lhd_{xy}, z_{xy})$, *and*
3. $(\lhd'_{xy}, z'_{xy}) + (\lhd_2, -d) < (\lhd_{x0}, z_{x0})$.

4.2.4 Algorithm for checking $Z \sqsubseteq_{G^{nd}} Z'$

Theorem 4.8 proved that the question of whether $Z \sqsubseteq_{G^{nd}} Z'$, given two canonical zones Z and Z', can be answered by checking if $Z \sqsubseteq_{\{\varphi\}} Z'$ holds for every $\varphi \in G^{nd}$ and if $Z \sqsubseteq_{\{x \lhd_1 c,\ d \lhd_2 y\}} Z'$ holds for every $x \lhd_1 c \in G^{nd}$ and $d \lhd_2 y \in G^{nd}$. Theorem 4.12 provided the conditions over the zones Z, Z' that are (necessary and) sufficient for concluding $Z \not\sqsubseteq_{\{x \lhd_1 c\}} Z'$, given a constraint $x \lhd_1 c$. Further, Theorem 4.16 stated the conditions that imply (and are implied by) $Z \not\sqsubseteq_{\{d \lhd_2 y\}} Z'$, given $d \lhd_2 y$. Lastly, Theorem 4.25 gave the conditions for $Z \not\sqsubseteq_{\{x \lhd_1 c,\ d \lhd_2 y\}} Z'$, given two constraints $x \lhd_1 c$ and $d \lhd_2 y$. The following theorem puts together these four theorems to get a consolidated set of conditions that are necessary as well as sufficient for $Z \not\sqsubseteq_{G^{nd}} Z'$.

Theorem 4.26. *Let Z, Z' be two non-empty canonical zones and $G^{nd} = G^u \cup G^\ell$ be a set of non-diagonal constraints, where G^u contains only upper-bound constraints and G^ℓ contains lower-bound constraints present in G^{nd}. Then, $Z \not\sqsubseteq_{G^{nd}} Z'$ iff at least one of the following three conditions hold:*

1. *$(\lhd'_{x0}, z'_{x0}) < (\lhd_{x0}, z_{x0})$ and $(\lhd_{x0}, z_{x0}) + (\lhd_1, c) \geq (\leq, 0)$ for some $x \lhd_1 c \in G^u$,*
2. *$(\lhd'_{0y}, z'_{0y}) < (\lhd_{0y}, z_{0y})$ and $(\lhd'_{0y}, z'_{0y}) + (\lhd_2, -d) < (\leq, 0)$ for some $d \lhd_2 y \in G^\ell$,*
3. *$(\lhd'_{xy}, z'_{xy}) < (\lhd_{xy}, z_{xy})$ and $(\lhd_{x0}, z_{x0}) + (\lhd_1, c) \geq (\leq, 0)$ and $(\lhd'_{xy}, z'_{xy}) + (\lhd_2, -d) < (\lhd_{x0}, z_{x0})$ for some $x \lhd_1 c \in G^u$ and $d \lhd_2 y \in G^\ell$.*

Proof. ($\Leftarrow$) If condition (1) holds then from Theorem 4.12 it follows that $Z \not\sqsubseteq_{\{x \lhd_1 c\}} Z'$ and therefore $Z \not\sqsubseteq_G Z'$ due to Theorem 4.8. Similarly, if condition (2) holds then Theorem 4.16 implies $Z \not\sqsubseteq_{\{d \lhd_2 y\}} Z'$, then Theorem 4.8 further implies that $Z \not\sqsubseteq_G Z'$. Lastly, if condition (3) holds then Theorem 4.25 implies $Z \not\sqsubseteq_{\{x \lhd_1 c,\ d \lhd_2 y\}} Z'$ and thus $Z \not\sqsubseteq_G Z'$, thanks to Theorem 4.8.

($\Rightarrow$) Assume $Z \not\sqsubseteq_{G^{nd}} Z'$. Theorem 4.8 implies that either $Z \not\sqsubseteq_{\{\varphi\}} Z'$ for some $\varphi \in G^{nd}$, or $Z \not\sqsubseteq_{\{x \lhd_1 c,\ d \lhd_2 y\}} Z'$, where $x \lhd_1 c \in G^{nd}$ and $d \lhd_2 y \in G^{nd}$. If $Z \not\sqsubseteq_{\{x \lhd_1 c\}} Z'$, then Theorem 4.12 implies that the condition (1) holds. Whereas, if $Z \not\sqsubseteq_{\{d \lhd_2 y\}} Z'$ then condition (2) follows from Theorem 4.16. Lastly, if $Z \not\sqsubseteq_{\{x \lhd_1 c,\ d \lhd_2 y\}} Z'$ is the case, then Theorem 4.25 implies condition (3). Therefore, at least one of the three conditions hold whenever $Z \not\sqsubseteq_{G^{nd}} Z'$. □

Theorem 4.26 translates to the following algorithm that checks if the relation $Z \sqsubseteq_{G^{nd}} Z'$ holds where G^{nd} is a set containing only non-diagonal constraints.

```
1  check Z ⊑_{G^nd} Z':
2      for every x ◁_1 c ∈ G^nd :
3          if (◁_{x0}, z_{x0}) + (◁_1, c) ≥ (≤, 0) and (◁'_{x0}, z'_{x0}) < (◁_{x0}, z_{x0}) :
4              return false
5      for every d ◁_2 y ∈ G^nd :
6          if (◁'_{0y}, z'_{0y}) + (◁_2, −d) < (≤, 0) and (◁'_{0y}, z'_{0y}) < (◁_{0y}, z_{0y}) :
7              return false
8      for every (x ◁_1 c, d ◁_2 y) ∈ G^u × G^ℓ :
9          if (◁_{x0}, z_{x0}) + (◁_1, c) ≥ (≤, 0) and (◁'_{xy}, z'_{xy}) + (◁_2, −d) < (◁_{x0}, z_{x0}) and
10         (◁'_{xy}, z'_{xy}) < (◁_{xy}, z_{xy}) :
11             return false
12     return true
```

Algorithm 5 Checking $Z \sqsubseteq_{G^{nd}} Z'$ where G^{nd} is a set of non-diagonal constraints

Maximum time taken by Algorithm 5 depends on the constraints present in the set G^{nd}. Each of the loops in Line 2 and Line 5 iterate over G^{nd} once and the loop in Line 8 iterates over every pair of $G^u \times G^\ell$. Since G^{nd} can be assumed to contain only two (one upper bound and one lower bound) constraints per clock (due to Corollary 3.6 and Corollary 3.10), Algorithm 5 takes at most $\mathcal{O}(|X|^2)$ time to complete in the worst case. This bound matches with the bound for the simulation relation $\sqsubseteq_{LU}$ shown by Herbreteau et al. in [HSW16]. Note that, since the time bound is dependent on the constraints present in G^{nd}, if G^{nd} contains much lesser number of constraints than $2|X|$, then the time taken by the algorithm can also potentially be much lesser than $\mathcal{O}(|X|^2)$. For example, if G^{nd} only contains lower

bound constraints, then the iteration in Line 8 will not be performed and the algorithm will take only linear time.

The next section will formulate an algorithm, using Algorithm 5, to check the relation $Z \sqsubseteq_G Z'$, when G will also be allowed to contain diagonal constraints.

4.3 Checking $Z \sqsubseteq_G Z'$ where G contains diagonals

Section 4.2 considered the relation $\sqsubseteq_{G^{nd}}$ where the set G^{nd} contained only non-diagonal constraints. This section considers diagonal constraints as well. The aim in this section is to develop an algorithm for checking the relation $Z \sqsubseteq_G Z'$, when G is a finite set of atomic constraints, that may also contain diagonal constraints.

Before focussing on the relation between zones, first a characterization is provided for the relation $v \sqsubseteq_G v'$, when G contains diagonal constraints. Recall from Definition 3.1 that $v \sqsubseteq_G v'$ holds if for every atomic constraint $\varphi \in G$ and for every constant $\delta \in \mathbb{R}_{\geq 0}$, $v + \delta \models \varphi$ implies $v' + \delta \models \varphi$. Now, since δ ranges over an uncountably infinite set it is difficult to ascertain whether $v \sqsubseteq_G v'$ or not. The following characterization states that the "for every constant $\delta \in \mathbb{R}_{\geq 0}$" can be removed for the diagonal constraints present in G. This is because: given a diagonal constraint φ, either $\forall \delta \in \mathbb{R}_{\geq 0}$, $v + \delta \models \varphi$, or $\forall \delta \in \mathbb{R}_{\geq 0}$, $v + \delta \not\models \varphi$.

Theorem 4.27. *Given a set of atomic constraints G, let $G^{nd} \subseteq G$ be the set of all non-diagonal constraints of G. Then $v \sqsubseteq_G v'$ iff the following two conditions hold:*

1. *for every diagonal constraint $\varphi \in G$, if $v \models \varphi$ then $v' \models \varphi$ as well, and*
2. *$v \sqsubseteq_{G^{nd}} v'$.*

Proof. ($\Rightarrow$) Assume $v \sqsubseteq_G v'$. Since $G^{nd} \subseteq G$, from Definition 3.1 it follows that $v \sqsubseteq_{G^{nd}} v'$. The first part also follows directly from Definition 3.1.

($\Leftarrow$) For the converse part, it needs to be shown that $v \sqsubseteq_G v'$. Note that, it is sufficient to show that, for every constraint $\varphi \in G$, $v \sqsubseteq_{\{\varphi\}} v'$. Choose a constraint $\varphi \in G$. If φ is a non-diagonal constraint, then $v \sqsubseteq_{\{\varphi\}} v'$ follows from the fact that $v \sqsubseteq_{G^{nd}} v'$. Now suppose φ is a diagonal constraint of the form $x - y \lhd c$ and $\delta \in \mathbb{R}_{\geq 0}$ is such that $v + \delta \models \varphi$. Now, $v + \delta \models x - y \lhd c \implies (v+\delta)(x) - (v+\delta)(y) \lhd c \implies v(x) + \delta - (v(y) + \delta) \lhd c \implies v(x) - v(y) \lhd c \implies v \models \varphi$. From the assumption, $v' \models \varphi$ as well. Then, following (backwards) the above sequence of implications, it can be derived that $v' + \delta \models \varphi$. Therefore, $v \sqsubseteq_{\{\varphi\}} v'$. Similar arguments also prove the case when φ is of the form $d \lhd x - y$. Hence, it follows that $v \sqsubseteq_G v'$. □

The above theorem can be reformulated in the following way: $v \sqsubseteq_G v'$ holds iff for every diagonal constraint $\varphi \in G$, (i) if $v \models \varphi$ then $v' \models \varphi$ and (ii) $v \sqsubseteq_{G^-} v'$, where $G^- = G \setminus \{\varphi\}$. By further unfolding the relation $v \sqsubseteq_{G^-} v'$, eventually all diagonal constraints can be taken outside of G^-, which then results in the formulation given in the theorem above. The following theorem is the extension of this (re-)formulation of Theorem 4.27 to the case of zones.

Theorem 4.28. *Let G be a finite set of atomic constraints and φ be a diagonal constraint present in G. Then, $Z \sqsubseteq_G Z'$ iff $Z \cap \varphi \sqsubseteq_{G^-} Z' \cap \varphi$ and $Z \cap \neg\varphi \sqsubseteq_{G^-} Z'$, where $G^- = G \setminus \{\varphi\}$.*

Proof. ($\Rightarrow$) Suppose $Z \sqsubseteq_G Z'$. Since $Z \cap \neg\varphi \subseteq Z$ clearly $Z \cap \neg\varphi \sqsubseteq_G Z'$ and hence by definition, $Z \cap \neg\varphi \sqsubseteq_{G^-} Z'$. For the other part, choose $v \in Z$ such that $v \models \varphi$. As $Z \sqsubseteq_G Z'$, there exists $v' \in Z'$ such that $v \sqsubseteq_G v'$. By definition, $v' \models \varphi$ since $\varphi \in G$ and $v \models \varphi$. Hence, $Z \cap \varphi \sqsubseteq_G Z' \cap \varphi$ and thereby $Z \cap \varphi \sqsubseteq_{G^-} Z' \cap \varphi$.

($\Leftarrow$) Suppose $Z \cap \varphi \sqsubseteq_{G^-} Z' \cap \varphi$ and $Z \cap \neg\varphi \sqsubseteq_{G^-} Z'$. Choose a valuation $v \in Z$. The goal is to find a valuation $v' \in Z'$ such that $v \sqsubseteq_G v'$. Now, either $v \in Z \cap \varphi$ or $v \in Z \cap \neg\varphi$. Suppose, $v \in Z \cap \varphi$. Then, since $Z \cap \varphi \sqsubseteq_{G^-} Z' \cap \varphi$, there exists $v' \in Z' \cap \varphi$ satisfying $v \sqsubseteq_{G^-} v'$. Firstly, note that both $v \models \varphi$ and $v' \models \varphi$. This observation coupled with $v \sqsubseteq_{G^-} v'$ give $v \sqsubseteq_G v'$. On the other hand, suppose $v \in Z \cap \neg\varphi$. Then, due to $Z \cap \neg\varphi \sqsubseteq_{G^-} Z'$, there is $v' \in Z'$ such that $v \sqsubseteq_{G^-} v'$. Since $v \not\models \varphi$ and $G = G^- \cup \{\varphi\}$, the relation $v \sqsubseteq_{G^-} v'$ implies the relation $v \sqsubseteq_G v'$. Hence in both the cases, there exists $v' \in Z'$ such that $v \sqsubseteq_G v'$, proving $Z \sqsubseteq_G Z'$. □

The above Theorem 4.28 reduces the problem of checking the relation $\sqsubseteq_G$ to two $\sqsubseteq_{G^-}$ checks, where G^- contains one less diagonal than G. These $\sqsubseteq_{G^-}$ checks can be recursively reduced to further checks with respect to sets containing lesser number of diagonals till G^- no longer contains any diagonal constraints and hence the check $\sqsubseteq_{G^-}$ becomes the check $\sqsubseteq_{G^{nd}}$. Algorithm 5 of Section 4.2 can then be used to check $\sqsubseteq_{G^{nd}}$. This translates to the following algorithm for checking $Z \sqsubseteq_G Z'$.

```
1 check Z ⊑_G Z':
2     if Z = ∅ :
3         return true
4     if Z' = ∅ :
5         return false
6     if Z ⋢_{G^nd} Z' :
7         return false
8     return Z ⊑*_G Z'
```

Algorithm 6

```
1 check Z ⊑*_G Z':
2     if G does not contain any diagonal
        constraints :
3         return true
4     pick a diagonal constraint
        φ = x − y ◁ c from G
5     G^- ⟵ G \ {φ}
6     if Z ∩ ¬φ ≠ ∅ :
7         if Z ∩ ¬φ ⋢*_{G^-} Z' :
8             return false
9     return Z ∩ φ ⊑_{G^-} Z' ∩ φ
```

Algorithm 7

The proof of correctness of the recursive algorithms described in Algorithm 6 and Algorithm 7, mainly follows from Theorem 4.28. However, the two algorithms use a few optimizations, the correctness of these are argued below. Lines 2 and 4 in Algorithm 6 consider two trivial cases for $Z \sqsubseteq_G Z'$: if Z is empty then $Z \sqsubseteq_G Z'$ and if $Z \neq \emptyset$ but $Z' = \emptyset$ then $Z \not\sqsubseteq_G Z'$ (these two results follow from Definition 3.4). The correctness of the optimization used in Line 6 of Algorithm 6 follows from the following proposition, which follows from Definition 3.4 and Proposition 3.2.

Proposition 4.29. *If $Z \not\sqsubseteq_{G^{nd}} Z'$ then $Z \not\sqsubseteq_G Z'$.*

Line 7 of Algorithm 7 uses another optimization: instead of checking $Z \cap \neg\varphi \sqsubseteq_{G^-} Z'$ it only checks $Z \cap \neg\varphi \sqsubseteq^*_{G^-} Z'$. This is because at this point, it is already known that $Z \cap \neg\varphi \neq \emptyset$, $Z' \neq \emptyset$. Also, since $Z \sqsubseteq_{G^{nd}} Z'$ and $Z \cap \neg\varphi \subseteq Z$, it follows that $Z \cap \neg\varphi \sqsubseteq_{G^{nd}} Z'$. Therefore, every **if** statement checked by Algorithm 6 before calling Algorithm 7 at Line 8, are already known to be false. Hence, these checks are not necessary and $\sqsubseteq^*_{G^-}$ can directly be checked. No more optimizations are present in the algorithms, the rest of Algorithm 7 is a translation of Theorem 4.28.

As discussed before, in Line 6 of Algorithm 6, Algorithm 5 can be used to test the relation $Z \sqsubseteq_{G^{nd}} Z'$. In the worst case, Algorithm 6 uses Algorithm 5 to test $Z \sqsubseteq_{G^{nd}} Z'$ exponentially many times in the number of diagonal constraints present in G and each $Z \sqsubseteq_{G^{nd}} Z'$ takes at most $\mathcal{O}(|X|^2)$ time, as discussed on Page 80. The next section provides a lower bound complexity for checking the relation $Z \sqsubseteq_G Z'$.

4.4 Complexity of checking $Z \not\sqsubseteq_G Z'$

Given two canonical zones Z, Z' and a finite set of atomic constraints G, Algorithm 6 on Page 82, in the worst case, may end up checking the relation $\sqsubseteq_{G^{nd}}$ exponentially (in the number of diagonals present in G) many times. The question is: are these exponentially many checks avoidable? This section proves that, checking whether $Z \not\sqsubseteq_G Z'$ is, in fact, NP-complete. The reduction given in this section is adapted from a similar reduction given for a slightly different relation in [GMS18].

Theorem 4.30. *Given two canonical zones Z, Z' and a finite set of atomic constraints G, checking if $Z \not\sqsubseteq_G Z'$ is* NP*-complete.*

The intuition behind this theorem comes from the following proposition.

Proposition 4.31. *Given a set of atomic constraints $G = G^{nd} \cup G^{diag}$, where G^{nd} is the set of all non-diagonal constraints and G^{diag} is the set of all diagonal constraints belonging to G. Then, $Z \not\sqsubseteq_G Z'$ iff $\exists S \subseteq G^{diag}$ such that –*

$$\left(Z \cap \left(\bigwedge_{\varphi \in S} \varphi\right) \cap \left(\bigwedge_{\varphi \in G^{diag} \setminus S} \neg\varphi\right)\right) \not\sqsubseteq_{G^{nd}} \left(Z' \cap \left(\bigwedge_{\varphi \in S} \varphi\right)\right)$$

Proof. The contrapositive statement of the proposition is proved below. In this proof for notational convenience, once the set S is fixed, let Z_{left} denote the zone $\left(Z \cap \left(\bigwedge_{\varphi \in S} \varphi\right) \cap \left(\bigwedge_{\varphi \in G^{diag} \setminus S} \neg\varphi\right)\right)$ and Z'_{right} denote $\left(Z' \cap \left(\bigwedge_{\varphi \in S} \varphi\right)\right)$.

($\Leftarrow$) Let $Z \sqsubseteq_G Z'$, then it is required to show that for every $S \subseteq G^{diag}$ the relation $Z_{left} \sqsubseteq_G Z'_{right}$ holds. Consider a valuation $v \in Z_{left}$. Since $Z \sqsubseteq_G Z'$, there exists a valuation $v' \in Z'$ such that $v \sqsubseteq_G v'$. Now, since $v \in Z_{left}$, for every diagonal constraint $\varphi \in S$, $v \models \varphi$. Since $v \sqsubseteq_G v'$, v' also satisfies each of the diagonals $\varphi \in S$. Therefore, $v' \in Z'_{right}$ and hence $Z_{left} \sqsubseteq_{G^{nd}} Z'_{right}$.

($\Rightarrow$) Assume for every $S \subseteq G^{diag}$ the relation $Z_{left} \sqsubseteq_{G^{nd}} Z'_{right}$ holds. Then, it is required to be proved that $Z \sqsubseteq_G Z'$. Consider a valuation $v \in Z$. Then, there exists a set of diagonals $S_v \subseteq G^{diag}$ such that $v \models \varphi$ for every $\varphi \in S_v$ and $v \not\models \varphi$ for every $\varphi \in G^{diag} \setminus S_v$. From the assumption it follows that there exists a valuation $v' \in Z'$ such that $v' \models \varphi$ for every $\varphi \in S_v$ and $v \sqsubseteq_{G^{nd}} v'$. Theorem 4.27 then implies that $v \sqsubseteq_G v'$. Therefore, the relation $Z \sqsubseteq_G Z'$ follows. □

Given a set of diagonal constraints $S \subseteq G$, whether the relation $\sqsubseteq_{G^{nd}}$ mentioned in the theorem holds or not can be checked in polynomial (in fact, in $\mathcal{O}(|X|^2)$) time. Moreover, since S contains diagonals already present in G, size of S is also polynomial in terms of the input. This implies that checking $Z \not\sqsubseteq_G Z'$ is in NP.

Lemma 4.32. *Given two canonical zones Z, Z' and a finite set of atomic constraints G, checking if $Z \not\sqsubseteq_G Z'$ is in* NP.

Proof. The problem belongs to NP because of the following:

- guess a set $S \subseteq G^{diag}$ (size of S is polynomial in the size of G),
- given such a set S whether the following non-relation holds or not can also be checked in polynomial time (discussed on Page 80) -

$$\left(Z \cap \left(\bigwedge_{\varphi \in S} \varphi\right) \cap \left(\bigwedge_{\varphi \in G^{diag} \setminus S} \neg\varphi\right)\right) \not\sqsubseteq_{G^{nd}} \left(Z' \cap \left(\bigwedge_{\varphi \in S} \varphi\right)\right)$$

□

The hardness comes from the difficulty in choosing the set S of Proposition 4.31, for which the relation $\sqsubseteq_{G^{nd}}$ does not hold. Finding out this S, may require to check all possible subsets of G^{diag}.

Checking $Z \not\sqsubseteq_G Z'$ is NP-hard.

The hardness is proved by showing a polynomial-time reduction from 3-SAT. Given a 3-CNF formula ϕ, the proof of hardness constructs two zones Z, Z' and a set of atomic constraints G such that ϕ is satisfiable iff $Z \not\sqsubseteq_G Z'$.

Notation. A *literal* is either a variable p or its negation $\neg p$, and a 3-*clause* is a disjunction of three literals $(l_1 \vee l_2 \vee l_3)$. A 3-CNF formula is a conjunction of 3-clauses. For a literal l, var(l) will denote the variable corresponding to l. For a 3-CNF formula ϕ, var(ϕ) will denote the variables present in ϕ. An *assignment* to a 3-CNF formula ϕ is a function from var(ϕ) to $\{\top, \bot\}$ ($\top$ denotes true and $\bot$ denotes false). For a clause C and an assignment σ, $\sigma \models C$ holds if substituting $\sigma(p)$ for each variable p occurring in C evaluates the clause to true. For a formula ϕ and an assignment σ, $\sigma \models \phi$ holds if all clauses of ϕ evaluate to true under σ. A formula ϕ is said to be *satisfiable* if there exists an assignment such that $\sigma \models \phi$. For the rest of the section, fix a 3-CNF formula $\phi := C_1 \wedge C_2 \wedge \cdots \wedge C_N$.

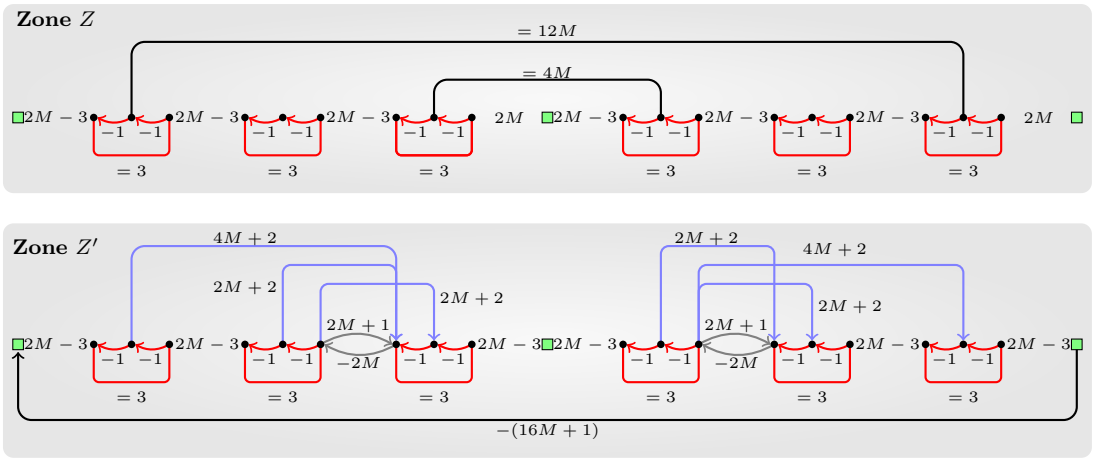


Figure 4.16: Illustration of the zone Z and Z' for the formula $(p_1 \vee p_2 \vee \neg p_3) \wedge (p_3 \vee \neg p_4 \vee \neg p_1)$. The separator clocks r_0, r_1, r_2 are shown by the green boxes (leftmost box is r_0, middle one is r_1 and the rightmost is r_2). The intermediate literal clocks are shown by the black dots: between r_0 and r_1 are $x_1^1, y_1^1, z_1^1, x_1^2, y_1^2, z_1^2, x_1^3, y_1^3, z_1^3$ in the same sequence. Similarly between r_1 and r_2 are the clocks $x_2^1, \ldots, z_2^3$. An edge of the form $x \xrightarrow{c} y$ simply denotes the constraint $y - x \leq c$, whereas edges $x \xrightarrow{=c} y$ mean that $y - x = c$. In some places, c is written in between two consecutive clocks, say $\star$ and $*$, to denote the edge $\star \xrightarrow{=c} *$, that is the value of $* - \star$ equals c.

The idea behind the reduction is the following. A formula ϕ is satisfiable iff $\exists$ an assignment σ such that $\forall$ clauses C of ϕ : $\sigma \models C$. Correspondingly, $Z \not\sqsubseteq_G Z'$ iff $\exists v \in Z$ such that $\forall\ v'$ satisfying $v \sqsubseteq_G v'$: $v' \notin Z'$. Given ϕ, the goal is to construct two zones Z, Z' and a set of atomic constraints G, such that ϕ is satisfiable iff $Z \not\sqsubseteq_G Z'$. The (potential) $v \in Z$ such that $v \notin {\downarrow}Z'$ (that is, every $v' \in Z'$ satisfies $v \not\sqsubseteq_G v'$) should encode the (potential) satisfying assignment for ϕ.

The zones Z, Z' that are to be constructed, consist of two kinds of valuations. One kind can be associated with assignments for the formula ϕ, the other cannot. The zone Z will be constructed in a way so that every assignment of ϕ has a corresponding valuation in Z. On the other hand, the zone Z' will be constructed to ensure that every valuation in Z' with which an assignment can be associated, falsifies at least one clause of the formula ϕ. Moreover, every assignment falsifying ϕ should have a corresponding valuation in Z'. Lastly, the set G will be constructed so that whenever $v \sqsubseteq_G v'$, for some valuation $v \in Z$ and $v' \in Z'$, if an assignment can be associated with v then the same assignment corresponds to the valuation v' as well. The detailed construction is described below.

For each literal l_i^j of ϕ, three clocks x_i^j, y_i^j, z_i^j are added. There are $N + 1$ additional clocks $r_0, r_1, \ldots, r_N$, where r_0 is assumed to be the special 0 clock, that is, every valuation maps the clock r_0 to the value 0. M is assumed to be an integer > 3. Figure 4.16 illustrates the construction.

The zone Z is described by three sets of constraints. The first set of constraints are between clocks of each literal. For every $i \in \{1, \ldots, N\}$ and $j \in \{1, 2, 3\}$:

$$y_i^j - x_i^j \geq 1 \quad \text{and} \quad z_i^j - y_i^j \geq 1 \quad \text{and} \quad z_i^j - x_i^j = 3 \tag{4.1}$$

The second set of constraints relates the distance between clocks of different literals. In addition, the clocks r_i are used as separators between clauses. For $i \in \{1, \ldots, N\}$:

$$x_i^1 - r_{i-1} = 2M - 3 \;\text{ and }\; x_i^{j+1} - z_i^j = 2M - 3 \text{ for } j \in \{1, 2\} \;\text{ and }\; r_i - z_i^3 = 2M \tag{4.2}$$

Constraints (4.1) and (4.2) and the fact that $2M - 3 >$, together ensure the following order of clocks for every valuation in Z, for each $i \in \{1, \ldots, N\}$:

$$r_{i-1} \;<\; x_i^1 < y_i^1 < z_i^1 \;<\; x_i^2 < y_i^2 < z_i^2 \;<\; x_i^3 < y_i^3 < z_i^3 \;<\; r_i \tag{4.3}$$

Note that for each $v \in Z$, the constraints of Z mentioned so far imply that $v(r_i) - v(r_{i-1}) = 8M$ for $i \in \{1, \ldots, N\}$. Now, in order for valuations in Z to represent a valid assignment to the variables of ϕ, the next set of constraints enforce that if ℓ_i^j and $\ell_{i'}^{j'}$ are two literals involving the same variable then the values of $y_i^j - x_i^j$ and $y_{i'}^{j'} - x_{i'}^{j'}$ are the same, for every valuation in Z. Without loss of generality, it can be assumed that the three literals in the same clause have different variables. Therefore, this condition is relevant for literals with the same variable in different clauses. For every l_i^j and $l_{i'}^{j'}$ such that $\text{var}(l_i^j) = \text{var}(l_{i'}^{j'})$ and $i' > i$:

$$y_{i'}^{j'} - y_i^j = (i' - i) \cdot 8M + (j' - j) \cdot 2M \tag{4.4}$$

Note that from (4.1) and (4.2) it can be inferred that the values of $v(x_{i'}^{j'}) - v(x_i^j)$ and $v(z_{i'}^{j'}) - v(z_i^j)$ are already equal to the right hand side of the above equation, as the x and z clocks are "fixed" and y is "flexible". Constraint (4.4) then ensures that $v(y_i^j) - v(x_i^j) = v(y_{i'}^{j'}) - v(x_{i'}^{j'})$ (and hence $v(z_i^j) - v(y_i^j) = v(z_{i'}^{j'}) - v(y_{i'}^{j'})$ as well) whenever l_i^j and $l_{i'}^{j'}$, with $i' > i$, have the same variable.

Encoding of assignments. By construction of Z, in every valuation $v \in Z$, $v(r_i)$, $v(x_i^j)$ and $v(z_i^j)$ are fixed integers. The clocks y_i^j are the only clocks whose values are not fixed. The value of $v(y_i^j)$ can vary between $v(x_i^j) + 1$ and $v(x_i^j) + 2$. When this value is in the extremes, either 1 or 2, the corresponding valuation, mapping every clock to an integer, is called an integral valuation. Assignments are encoded by such integral valuations. An integral valuation v encodes the assignment σ_v where:

$$\sigma_v(\text{var}(l_i^j)) = \begin{cases} \top & \text{if } v(y_i^j) - v(x_i^j) = 1 \\ \bot & \text{if } v(y_i^j) - v(x_i^j) = 2 \end{cases} \tag{4.5}$$

By (4.4), the above assignment is well defined. Moreover, the zone Z contains an integral valuation for every possible assignment.

Lemma 4.33. *For every assignment σ to the variables of ϕ, the zone Z described by (4.1), (4.2) and (4.4) contains an integral valuation v such that the assignment σ_v as defined in (4.5) equals the assignment σ.*

An assignment σ satisfies ϕ if every clause evaluates to true under σ. From a valuation v encoding this assignment σ, a mechanism is required to check whether each clause is true. The differences between the clocks x_i^j and z_i^{j-1} or r_{i-1} will be used for this purpose. Clauses will be identified by certain kind of shifts to these differences in v. Some more notations are introduced here. Let $L := \{(x_i^j, y_i^j, z_i^j) \mid i \in \{1, \ldots, N\} \text{ and } j \in \{1, 2, 3\}\}$ be the triplets of clocks associated with each literal. A literal is said to be *positive* if it is a variable p, and it is *negative* if it is the negation $\neg p$ of some variable p. It will be assumed that in every clause of ϕ, the positive literals are written before the negative literals: for example, $p_1 \vee p_3 \vee \neg p_2$ will be written instead of $p_1 \vee \neg p_2 \vee p_3$. For each clause C_i, let (e_i, f_i) be the pair of clocks corresponding to C_i in the *border* between positive and negative literals:

$$(e_i, f_i) := \begin{cases} (r_{i-1}, x_i^1) & \text{if all literals in } C_i \text{ are negative} \\ (z_i^j, x_i^{j+1}) & \text{if for } j \in \{1,2\},\ l_i^j \text{ is positive and } l_i^{j+1} \text{ is negative} \\ (z_i^3, r_i) & \text{if all literals in } C_i \text{ are positive} \end{cases} \tag{4.6}$$

Given the formula ϕ, the above border clocks are fixed. The zone Z' will allow the difference in between these border clocks to be 'flexible', in between $2M$ and $2M+1$. All the other constraints on the differences between any two consecutive clocks remains the same as that of Z. A valuation v' of Z' will be said to *activate* a clause C_i of ϕ, if for v' the difference $v'(f_i) - v'(e_i) = 2M + 1$. Z' will be constructed in a way that ensures every valuation of Z' activates at least one clause of ϕ. Moreover, if v' is an integral valuation encoding the assignment $\sigma_{v'}$ (according to 4.5) then for every clause C_i that v' activates, $\sigma_{v'} \not\models C_i$.

The zone Z′ is described by five sets of constraints. The first set of constraints are between the clocks of the same literal, and are identical to that in Z:

$$y_i^j - x_i^j \geq 1 \quad \text{and} \quad z_i^j - y_i^j \geq 1 \quad \text{and} \quad z_i^j - x_i^j = 3 \tag{4.7}$$

The second set of constraints are for border clocks (these are already determined given the formula ϕ) in each clause. For each $i \in \{1, \ldots, N\}$:

$$2M \leq f_i - e_i \leq 2M + 1 \tag{4.8}$$

where e_i and f_i are according to the definition in (4.6). The third set of constraints fix differences between consecutive blocks not involving border clocks to $2M - 3$.

$$\begin{aligned} x_i^1 - r_{i-1} &= 2M - 3 \quad \text{if } (r_{i-1}, x_i^1) \neq (e_i, f_i) \text{ and} \\ x_i^{j+1} - z_i^j &= 2M - 3 \quad \text{for } j \in \{1,2\} \text{ when } (z_i^j, x_i^{j+1}) \neq (e_i, f_i) \text{ and} \\ r_i - z_i^3 &= 2M - 3 \quad \text{when } (z_i^3, r_i) \neq (e_i, f_i) \end{aligned} \tag{4.9}$$

From (4.7, 4.8, 4.9), note that for every valuation in Z' the difference $r_i - r_{i-1}$ is between $8M$ and $8M+1$ with the flexibility coming from $f_i - e_i$. The fourth set of constraints ensures that at least one of the $f_i - e_i$ should be bigger than $2M$.

$$r_N - r_0 \geq (8M \cdot N) + 1 \tag{4.10}$$

So far, the constraints that have been chosen for Z' do not talk about any clauses being true or false. Recall that valuation v is said to activate a clause C_i if the border clocks satisfy $v(f_i) - v(e_i) = 2M + 1$. The final set of constraints ensures that whenever an integral valuation v' in Z' activates a clause C_i, that is, $v'(f_i) - v'(e_i) = 2M + 1$, every literal in C_i evaluates to false under the encoding scheme given in (4.5): if l_i^j is positive then $v'(y_i^j) - v'(x_i^j) = 2$ and if l_i^j is negative then $v'(y_i^j) - v'(x_i^j) = 1$. For a positive literal l_i^j let $d_i^j \in \{0, 1, 2\}$ be the number of (x, y, z) blocks corresponding to positive literals between z_i^j and f_i (does not include j). Similarly, for a negative literal, let $d_i^j \in \{0, 1, 2\}$ be the number of blocks corresponding to negative literals between e_i and x_i^j (again, excludes j). Lastly, Z' contains the following constraints:

$$\begin{aligned} f_i - y_i^j &\leq d_i^j \cdot 2M + (2M + 2) \quad \text{if } l_i^j \text{ is a positive literal} \\ y_i^j - e_i &\leq d_i^j \cdot 2M + (2M + 2) \quad \text{if } l_i^j \text{ is a negative literal} \end{aligned} \tag{4.11}$$

Note that, not every integral valuation in Z' will correspond to a valid assignment to the variables of ϕ. This is because the constraints in (4.4) that were enforced in Z are not enforced in Z'. An integral valuation will be called an *assignable* valuation if it corresponds to a valid assignment (with respect to (4.5)).

Lemma 4.34. *For every integral valuation $v' \in Z'$, if v' is assignable then $\sigma_{v'} \not\models \phi$.*

Proof. Let v' be an assignable integral valuation in Z'. Let $\sigma_{v'}$ be the assignment corresponding to v' according to the encoding scheme presented in (4.5). Because of the constraint in (4.10) v' activates at least one clause, say $C_i = \ell_1 \vee \ell_2 \vee \ell_3$. For every $i \in \{1, 2, 3\}$, if ℓ_i is positive (resp. negative) then because of the constraints in (4.11) it follows that $v'(y_i) - v'(x_i) = 2$ (resp. $v'(y_i) - v'(x_i) = 1$) and therefore $\sigma_{v'}(\text{var}(\ell_i)) = \bot$ (resp. $\sigma_{v'}(\text{var}(\ell_i)) = \top$). Hence, $\sigma_{v'} \not\models C_i$ and $\sigma_{v'} \not\models \phi$. □

The last remaining part of the reduction is the construction of the set G. The aim behind this construction is to ensure that whenever an integral valuation $v \in Z$ (which corresponds to the assignment σ_v) satisfies $v \sqsubseteq_G v'$ with v' being a valuation in Z', v' should be assignable and moreover $\sigma_v = \sigma_{v'}$ should hold as well.

The set G consists of the following diagonal constraints: assuming the formula ϕ consists of N clauses, the constraints $y_i^j - x_i^j \leq 1$ and $y_i^j - x_i^j \geq 2$ belong to G, for every $i \in \{1, 2, 3, \ldots, N\}$ and for every $j \in \{1, 2, 3\}$.

The lemma below proves that the aim behind the construction of G is fulfilled.

Lemma 4.35. *Let v be an integral valuation in Z and v' be a valuation in Z'. Then, $v \sqsubseteq_G v'$ holds iff - (i) v' is an assignable valuation and (ii) $\sigma_v = \sigma_{v'}$.*

Proof. ($\Rightarrow$) Since $v \in Z$ is an integral valuation, for every $i \in \{1, 2, \ldots, N\}$ and every $j \in \{1, 2, 3\}$ the value of $v(y_i^j) - v(x_i^j)$ is either 1 or 2. Suppose $v(y_i^j) - v(x_i^j) = 1$ (resp. $v(y_i^j) - v(x_i^j) = 2$). Then, $v \models y_i^j - x_i^j \leq 1$ (resp. $v \models y_i^j - x_i^j \geq 2$). Since $v \sqsubseteq_G v'$ it then follows that $v' \models y_i^j - x_i^j \leq 1$ (resp. $v' \models y_i^j - x_i^j \leq 2$). From the construction of the zones Z and Z' (the constraints in (4.1) and (4.7)) it follows that $1 \leq v'(y_i^j) - v'(x_i^j) \leq 2$. Therefore, $v'(y_i^j) - v'(x_i^j) = 1$ (resp. $v'(y_i^j) - v'(x_i^j) = 2$)

as well. Then, in fact $v'(y_i^j) - v'(x_i^j) = v(y_i^j) - v(x_i^j)$. Since this equality holds for every i and j, and since v is assignable both of the results follow. That is, firstly v' is also assignable. Moreover, $v(y_i^j) - v(x_i^j) = v'(y_i^j) - v'(x_i^j)$ implies that $\sigma_v = \sigma_{v'}$.

($\Leftarrow$) Let $v' \in Z'$ be an assignable valuation such that $\sigma_v = \sigma_{v'}$. This implies that $v(y_i^j) - v(x_i^j) = v'(y_i^j) - v'(x_i^j)$ holds for every $i \in \{1, 2, 3, \ldots, N\}$ and $j \in \{1, 2, 3\}$. Since the constraints in G only involve the difference $y_i^j - x_i^j$, $v \sqsubseteq_G v'$ holds. □

The following lemma proves that for every assignment σ falsifying ϕ, the zone Z' contains an assignable valuation v' such that $\sigma_{v'}$ is the assignment σ.

Lemma 4.36. *Let σ be an assignment such that $\sigma \not\models \phi$. Then, Z' contains an assignable valuation v' such that $\sigma_{v'} = \sigma$.*

Proof. Given an assignment σ such that $\sigma \not\models \phi$, this proof constructs an assignable valuation $v' \in Z'$ such that $\sigma_{v'} = \sigma$. Since for every valuation in Z' the value of the clock r_0 is 0, first set $v'(r_0) = 0$ as well. Now, for every clause C_i of ϕ such that $\sigma \not\models C_i$, the difference between the border clocks (as mentioned in (4.6)) should be $2M + 1$ for v' and for all the other clauses the difference between its border clocks under v' should be $2M$. This convention and the constraints in (4.7) and (4.9) fix the values of every clock r, x, z under v'. The remaining task is to assign the values $v'(y)$ for the clocks y. This is fixed based on the assignment σ. For every $i \in \{1, 2, 3, \ldots, N\}$ and every $j \in \{1, 2, 3\}$, set $v'(y_i^j)$ such that: $v'(y_i^j) - v'(x_i^j) = 1$ if $\sigma(\text{var}(\ell_i^j)) = \top$ and $v'(y_i^j) - v'(x_i^j) = 2$ if $\sigma(\text{var}(\ell_i^j)) = \bot$. This implies that v' is assignable and $\sigma_{v'} = \sigma$. Now, the claim is the valuation v' belongs to the zone Z'. Since the values of the clocks r, x, z are determined by the constraints of Z', the constraints in (4.7) and (4.9) are satisfied by v'. The constraints over the border clocks presented in (4.8) is also satisfied since v' ensures these distances are either $2M$ or $2M + 1$. Also, since $\sigma \not\models \phi$, there is at least one clause C_i such that $\sigma \not\models C_i$. Then from the construction of v', $v'(f_i) - v'(e_i) = 2M + 1$. This ensures v' satisfies the constraint in (4.10). Only the satisfaction of the constraints in (4.11) now remains to be shown. It can be checked that the constraints in (4.11) imply that for every valuation in Z', if the difference $f_i - e_i$ is $2M + 1$ then for every $j \in \{1, 2, 3\}$ the value of the difference $y_i^j - x_i^j$ is 2 for positive literals and 1 for negative literals. On the other hand, if the difference $f_i - e_i$ is $2M$ instead, then $y_i^j - x_i^j$ can be any value in between 1 and 2. From the construction of v', the value of $f_i - e_i$ is $2M + 1$ only for the clauses C_i where $\sigma \not\models C_i$. For every literal ℓ_i^j of this clause, $\sigma(\ell_i^j) = \bot$. Then $v'(y_i^j) - v'(x_i^j)$ will indeed be 2 if ℓ_i^j is positive and 1 otherwise. This implies that for these clauses also v' satisfies the constraints of (4.11). Therefore, the valuation v' belongs to the zone Z'. □

The following lemma proves that if $Z \not\sqsubseteq_G Z'$ then Z contains an integral valuation that serves as the certificate for $Z \not\sqsubseteq_G Z'$.

Lemma 4.37. *If $Z \not\sqsubseteq_G Z'$ then $\exists$ an integral valuation $v \in Z$ such that $v \notin \downarrow Z'$.*

Proof. Assume $Z \not\sqsubseteq_G Z'$. Let $u \in Z$ be such that $u \notin \downarrow Z'$. If u is integral then the lemma is proved. Now, assume u is not an integral valuation. Recall that, for all the clocks r, x, z the values are fixed by the zone Z and these are integers. Only the

values for the clocks y are flexible. Therefore, since u is non-integral, there exists i, j such that $u(y_i^j)$ is not an integer. This proof constructs an integral valuation $v \in Z$ based on u, such that $v \notin \downarrow Z'$ as well.

The valuation v is constructed in the following way. The values of the clocks x, y and r are already fixed from Z. Only the values of the clocks y need to be fixed. For every $i \in \{1, 2, 3, \ldots, N\}$ and $j \in \{1, 2, 3\}$, set the value of $v(y_i^j)$ so that:

$$v(y_i^j) - v(x_i^j) = \begin{cases} u(y_i^j) - u(x_i^j) & \text{if } u(y_i^j) - u(x_i^j) \in \mathbb{Z} \\ 1 & \text{if } u(y_i^j) - u(x_i^j) \notin \mathbb{Z} \end{cases}$$

From the construction of v, firstly, v is integral. It needs to be shown that $v \in Z$. Now, v satisfies the constraints in (4.1), since u does and the constraints in (4.2) because of the construction of v. Now, let l_i^j and $l_{i'}^{j'}$ be two literals containing the same variable, that is, $\text{var}(l_i^j) = \text{var}(l_{i'}^{j'})$. Then, it needs to be shown that $v(y_i^j) - v(x_i^j) = v(y_{i'}^{j'}) - v(x_{i'}^{j'})$. There are two possibilities: (i) either the value $u(y_i^j) - u(x_i^j)$ is an integer and in this case, $v(y_i^j) - v(x_i^j) = u(y_i^j) - u(x_i^j)$ as well as $v(y_{i'}^{j'}) - v(x_{i'}^{j'}) = u(y_{i'}^{j'}) - u(x_{i'}^{j'})$ and $u(y_i^j) - u(x_i^j) = u(y_{i'}^{j'}) - u(x_{i'}^{j'})$ since $u \in Z$, otherwise (ii) $u(y_{i'}^{j'}) - u(x_{i'}^{j'})$ is not an integer. Now, since $u \in Z$, $u(y_i^j) - u(x_i^j) = u(y_{i'}^{j'}) - u(x_{i'}^{j'})$ holds as well. Therefore, in this case, from the construction of v, $v(y_i^j) - v(x_i^j) = 1 = v(y_{i'}^{j'}) - v(x_{i'}^{j'})$. Thus in both of the cases it follows that v satisfies the constraints of (4.4) as well. Therefore, $v \in Z$.

It will now be proved that if $v \in \downarrow Z'$ then $u \in \downarrow Z'$ as well. Assume $v \in \downarrow Z'$. Then, there exists $v' \in Z'$ such that $v \sqsubseteq_G v'$. The claim now is that $u \sqsubseteq_G v'$ also holds. Choose a constraint $\varphi := y_i^j - x_i^j \bowtie c \in G$. It needs to be shown that $u \sqsubseteq_{\{\varphi\}} v'$. From the construction of G, it follows that $(\bowtie, c)$ is either $(\leq, 1)$ or $(\geq, 2)$. Now, if $u(y_i^j) - u(x_i^j)$ is an integer then from the construction of v, $v(y_i^j) - v(x_i^j) = u(y_i^j) - u(x_i^j)$. Therefore, if $u \models \varphi$ then $v \models \varphi$. Since $v \sqsubseteq_G v'$, $v' \models \varphi$ as well, proving that $u \sqsubseteq_{\{\varphi\}} v'$. Otherwise, if $u(y_i^j) - u(x_i^j)$ is not an integer, then, $u \not\models \varphi$. Hence, in this case $u \sqsubseteq_{\{\varphi\}} v'$ holds trivially. Therefore, $u \sqsubseteq_G v'$.

Now, if $v \in \downarrow Z'$ then from the argument above $u \in \downarrow Z'$ as well. But this contradicts the assumption that $u \notin \downarrow Z'$. Therefore, $v \notin \downarrow Z'$ must also hold. □

The following lemma is the final step towards showing the NP-hardness.

Lemma 4.38. *Formula ϕ is satisfiable iff $Z \not\sqsubseteq_G Z'$.*

Proof. ($\Rightarrow$) Assume the given 3-CNF formula ϕ is satisfiable. Then, there exists an assignment σ such that $\sigma \models \phi$. It needs to be shown that with the zones Z, Z' and the set G constructed, the relation $Z \not\sqsubseteq_G Z'$ holds. From Lemma 4.33 it follows that Z contains a valuation v such that $\sigma = \sigma_v$. The claim is $v \notin \downarrow Z'$. Let $v' \in Z'$ be a valuation such that $v \sqsubseteq_G v'$. From Lemma 4.35 it follows that $\sigma_v = \sigma_{v'}$. Since, $\sigma \models \phi$ and $\sigma = \sigma_v = \sigma_{v'}$, $\sigma_{v'} \models \phi$ as well. But this contradicts Lemma 4.34.

($\Leftarrow$) Conversely, suppose $Z \not\sqsubseteq_G Z'$. Lemma 4.37 implies that Z contains an integral valuation v such that $v \notin \downarrow Z'$. The claim is $\sigma_v \models \phi$. Suppose this is not true, that is, $\sigma_v \not\models \phi$. Then, Lemma 4.36 implies that Z' contains a valuation v' such that $\sigma_{v'} \not\models \phi$ and $\sigma_v = \sigma_{v'}$. The latter result together with Lemma 4.35 implies that $v \sqsubseteq_G v'$, which contradicts the fact that $v \notin \downarrow Z'$. □

The lemma above implies the NP-hardness.

Lemma 4.39. *Given two canonical zones Z, Z' and a finite set of constraints G, checking if $Z \not\sqsubseteq_G Z'$ is* NP*-hard.*

Lemma 4.39 together with Lemma 4.32 proves that checking $Z \not\sqsubseteq_G Z'$ is NP-complete (Theorem 4.30).

4.5 Discussion

The goal in this chapter was to devise an algorithm for checking if the relation $Z \sqsubseteq_G Z'$ holds or not, given two zones Z, Z' and a finite set of atomic constraints G. An algorithm for checking the relation $Z \sqsubseteq_G Z'$ when G does not contain diagonal constraints was presented in Section 4.2 and an algorithm when G is allowed to contain diagonal constraints was discussed in the following Section 4.3.

For the diagonal-free case, there already exists the LU simulation relation, which can be checked in quadratic (in the number of clocks present in the underlying automaton) time. The time to decide $\sqsubseteq_G$ in the diagonal-free case is also at most quadratic, however, this complexity actually depends on the number of upper and lower bound (non-diagonal) constraints present in G. If G contains only lower-bound (or only upper-bound) non-diagonal constraints, then the relation $\sqsubseteq_G$ can, in fact, be checked in linear time. The complexity approaches the upper bound when G contains many lower as well as many upper bound constraints.

The algorithm for checking $Z \sqsubseteq_G Z'$ when G is allowed to contain diagonal constraints, makes use of the algorithm present for the diagonal-free case. In the worst case, it might be required to call the diagonal-free algorithm exponentially many (in the number of diagonal constraints present in G) times. However, it might be possible to conclude whether $Z \sqsubseteq_G Z'$ holds or not with much fewer such checks in certain cases. Instead of using the diagonal-free version of the $\sqsubseteq_G$ relation, the LU simulation relation can also be used to conclude when $Z \sqsubseteq_G Z'$ holds. Although this would be sound, it only computes an approximation of $\sqsubseteq_G$ and therefore may end up concluding $Z \sqsubseteq_G Z'$ does not hold in certain cases where it actually holds.

Section 4.4 proved that deciding $Z \sqsubseteq_G Z'$ when G contains diagonal constraints is NP-complete. Therefore, instead of using the algorithm presented in Section 4.3, the problem $Z \sqsubseteq_G Z'$ can also be encoded as a formula in a suitable theory and then decided by using satisfiability solvers for that theory. An attempt at this was made for an earlier version of the relation $\sqsubseteq_G$ (deciding which was also NP-complete) in the work [GMS18], but it did not perform well in terms of total execution time. However, finding better encoding is one possible direction to be investigated further.

Section 4.1 proved that the relation $\sqsubseteq_G$ is finite, whenever G is a finite set of atomic constraints and whenever $\sqsubseteq_G$ is finite, the reachability algorithm using this $\sqsubseteq_G$ is guaranteed to terminate. Chapter 3 provided a specific construction for this set G that makes the relation $\sqsubseteq_G$ a simulation relation for Updatable Timed Automata. Recall that this G is not always finite given an updatable timed automaton. However, every automaton for which this G is finite, the reachability algorithm using the relation $\sqsubseteq_G$ can be used successfully to check if a given state of the input

automaton is reachable or not. The decidability proofs for subclasses of Updatable Timed Automata use the region construction. The results in this chapter and the previous, provide an alternative way of proving decidability. If it can be proved that for a subclass of Updatable Timed Automata, the reduced $\mathcal{A}$-map computation (Algorithm 4) always terminates with a finite reduced $\mathcal{A}$-map, reachability can be decided for that class of automata. The next chapter proves decidability for certain classes of Updatable Timed Automata using this way.

Chapter 5

Applications of the simulation relation $\sqsubseteq_{\mathcal{G}}$

Chapter 3 defined the relation $\sqsubseteq_G$, made it a simulation relation for Updatable Timed Automata and Chapter 4 described an algorithm for checking $\sqsubseteq_G$ between two zones. This algorithm can be used to check reachability in every updatable timed automata for which the reduced $\mathcal{A}$-map computation (Algorithm 4) terminates. The question now is for which automata do Algorithm 4 terminate? This chapter presents some of the known subclasses of Updatable Timed Automata for which reachability is decidable and proves that for all those classes the reduced $\mathcal{A}$-map computation terminates, enabling the use of the algorithm discussed in Chapter 4. This chapter concludes with a modified reachability problem, which can also be decided using the algorithm in Chapter 4.

5.1 Subclasses with decidable reachability

In [BDFP04], Bouyer et al. detailed a list of subclasses[1] of Updatable Timed Automata for which the reachability problem is decidable. This section proves that for all these classes of automata, the reduced $\mathcal{A}$-map computation terminates and for all the undecidable classes there are automata for which the reduced $\mathcal{A}$-map computation does not terminate. This list of decidability results are recalled in Table 5.1. In this table c and d are integers with $c > 0$ and $d \geq 0$.

Theorem 5.1. *Given an updatable timed automaton $\mathcal{A}$, if $\mathcal{A}$ belongs to a subclass of Updatable Timed Automata for which reachability is decidable according to Table 5.1, then the reduced $\mathcal{A}$-map computation terminates.*

Proof. Given an atomic constraint φ and an update up of form $x := y$, the constraint $\mathsf{pre}(\varphi, g, up)$ contains the same constant as φ. Therefore, this update alone cannot

[1] Along with the updates mentioned in Table 5.1, [BDFP04] also considers non-deterministic updates of the form $x :< c, x :< y + d$, etc.. This thesis only considers the deterministic updates.

$\{x := 0\} \cup$	without diagonals	containing diagonals
$x := d,\ x := y$	decidable	decidable
$x := x + c$	decidable	undecidable
$x := x - c$	undecidable	undecidable

Table 5.1: Subclasses of UTA with decidable reachability

generate infinitely many constraints and hence the reduced $\mathcal{A}$-map computation terminates. On the other hand, with up being of the form $x := d$, $\mathsf{pre}(\varphi, g, up)$ is a trivial constraint when φ is a non-diagonal constraint and when φ is a diagonal constraint, $\mathsf{pre}(\varphi, g, up)$ may contain a different constant than φ but the diagonal constraint becomes a non-diagonal constraint and therefore, again, it cannot result in infinitely many propagations. Therefore, for both of the classes in Row 1 of Table 5.1, the reduced $\mathcal{A}$-map computation terminates.

Now consider the update $x := x + c$ for some integer $c > 0$. The decidability holds only when the automaton is diagonal-free. For a non-diagonal constraint φ of the form $x \triangleleft c_x$ or $c_x \triangleleft x$, when $up_x = x + c$, the constraint $\mathsf{pre}(\varphi, g, up)$ contains the constant $c_x - c$, which is strictly smaller than c_x. Since the constants in constraints needs to be non-negative, this update can again not result in infinite propagations. Therefore, the reduced $\mathcal{A}$-map computation terminates. □

For the undecidable classes, there are examples for which the reduced $\mathcal{A}$-map computation does not terminate. This is expected, since termination of reduced $\mathcal{A}$-map computation implies decidability. For each of the examples mentioned below, if $\mathcal{G}$ is its corresponding reduced $\mathcal{A}$-map, then the set $\mathcal{G}(q_0)$ becomes infinite resulting in non-termination of the reduced $\mathcal{A}$-map computation.

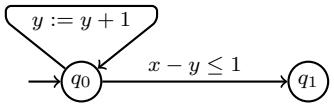
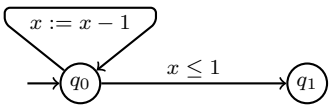
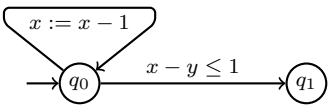


Figure 5.1: Non-terminating reduced $\mathcal{A}$-map: update of the form $x := x + c$ and diagonal constraints

For the automaton in Figure 5.1, in the reduced $\mathcal{A}$-map $\mathcal{G}$, the set $\mathcal{G}(q_0) = \{x - y \leq 1,\ x - y \leq 2,\ x - y \leq 3,\ \dots\}$. Intuitively, additive updates $x := x + c$ have the effect of subtraction updates, when diagonal constraints are present and therefore the problem becomes undecidable in the presence of diagonals.



(a) no diagonal constraints



(b) contains diagonal constraints

Figure 5.2: Non-terminating reduced $\mathcal{A}$-map: update of the form $x := x - c$

For the automaton in Figure 5.2a, the set $\mathcal{G}(q_0) = \{x \leq 1,\ x \leq 2,\ x \leq 3,\ \dots\}$. For the automaton in Figure 5.2b, $\mathcal{G}(q_0) = \{x - y \leq 1, x - y \leq 2, x - y \leq 3, \dots\}$.

5.2 Timed Automata with Bounded Subtraction

Checking reachability in Updatable Timed Automata with updates being only of the form $x := 0$ (reset) and $x := x - c$ (subtraction update), where $c \in \mathbb{N}$, is known to be undecidable [BDFP04]. However, in order to model preemptive scheduling, a decidable subclass of this was introduced by Fersman et al. in [FKPY07], called *Timed Automata with Bounded Subtraction*. The decidability was proved by using a modified region automaton construction, this section provides an alternate proof of decidability through the finiteness of reduced $\mathcal{A}$-maps.

The only restriction present in this subclass is: whenever a transition with a subtraction update ($x := x - c$, x is a clock and $c \in \mathbb{N}$) is enabled from a *reachable* configuration (q, v), the value $v(x)$ must be bounded by a *known* constant. The following definition provides the formal syntax of this class of automata.

Definition 5.2 (Timed Automata with Bounded Subtraction [FKPY07])**.** *A timed automaton with "subtraction" is an updatable timed automaton with updates restricted to the form $x := 0$ and $x := x - c$ for $c \in \mathbb{N}$. The guards may contain both diagonal and non-diagonal constraints and their conjunctions.*

A timed automaton with subtraction is called a timed automaton with "bounded subtraction" if there exists a constant M_x for every clock x, such that, for every reachable configuration (q, v) of the automaton: if there exists a transition (q, g, up, q') such that $up_x = x - c$ with $c \in \mathbb{N}$ and if $v \models g$, then $v(x) \leq M_x$.

Theorem 5.3 ([FKPY07])**.** *Given a timed automaton with bounded subtraction, if the bounds M_x are known for every clock x, then, checking reachability is decidable.*

As examples, the automaton in Figure 3.5 is a timed automaton with subtraction, whereas, the automaton in Figure 3.3 is a timed automaton with bounded subtraction. The reduced $\mathcal{A}$-map construction does not terminate for the former while it does terminate for the latter. However, even for timed automata with bounded subtraction, the reduced $\mathcal{A}$-map computation may not terminate.

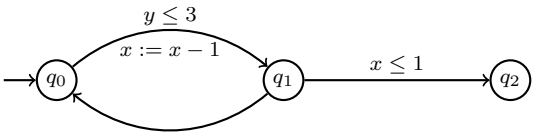


Figure 5.3: Non-terminating reduced $\mathcal{A}$-map computation for a timed automaton with bounded subtraction with the bounds $M_x = 3$ and $M_y = 3$

The reduced $\mathcal{A}$-map computation of Algorithm 3 does not terminate for the automaton in Figure 5.3. This is because of the constraint $x \leq 1$ present in the transition from q_1 to q_2. This constraint $x \leq 1$ first gets added to the set $\mathcal{G}(q_1)$ and then propagates to $\mathcal{G}(q_0)$ as the constraint $\mathsf{pre}(x \leq 1, y \leq 3, x := x - 1) = x \leq 2$. As the computation progresses, the constraints $x \leq i$ for $i \in \mathbb{N}_{\geq 1}$ get added to the set $\mathcal{G}(q_0)$ and therefore the fixpoint computation fails to terminate. Note that, for this automaton the value of the constant N described in Proposition 3.37 is $\max(3, 1) + (2 \times 1 \times 3 \times 2^2) = 27$. Since the constraint $x \leq 28$ will eventually get

added to $\mathcal{G}(q_0)$, Algorithm 4 will be able to conclude that the fixpoint computation does not terminate for this automaton.

In the automaton of Figure 5.3, for every reachable configuration (q_i, v), where $i \in \{0, 1, 2\}$, it is always true that $v(x) \leq v(y)$. This is because the only update present in the automaton, reduces the value of x. Now, every configuration (q_0, v) from where the transition $(q_0, y \leq 3, x := x - 1, q_1)$ is enabled, $v(x) \leq v(y) \leq 3$. Therefore $M_x = 3$ serves as the bound for the clock x and since $y \leq 3$ is present in the guard of this transition – the only transition with a subtraction update – the value $M_y = 3$ also serves as the bound for the clock y. Therefore, the automaton in Figure 5.3 is a timed automaton with bounded subtraction.

The bounds on the clocks imposed by Definition 5.2 are semantic in nature. Therefore, these do not affect the $\mathcal{A}$-map construction. It is possible to define a modified class of automata, that further puts a syntactic bound on the clocks, keeping it equally expressive as timed automata with bounded subtraction with known bounds and for which the reduced $\mathcal{A}$-map computation is guaranteed to terminate.

Definition 5.4 (Timed Automata with Syntactically Bounded Subtraction)**.** *A timed automaton with syntactically bounded subtraction is a timed automaton with subtraction, such that, for every transition (q, g, up, q') and clock x, if $up_x = x - c$ for some $c \in \mathbb{N}$ then the guard g contains an upper bound constraint $x \triangleleft c'$ for some $c' \in \mathbb{N}$.*

The following result states that every timed automaton with bounded subtraction can be modified into a timed automaton with syntactically bounded subtraction, preserving the runs of the automaton. The proof follows from the following observation: for every transition $t = (q, g, up, q')$ containing a subtraction update, for every reachable valuation v satisfying the guard g, for every clock x, the relation $v(x) \leq M_x$ holds. Therefore, every such valuation will also satisfy the (new) guard $g \wedge x \leq M_x$. This construction relies on the fact that the bounds M_x are known.

Lemma 5.5. *For every timed automaton with bounded subtraction $\mathcal{A}'$ where the bounds M_x for every clock x are known, there exists a timed automaton with syntactically bounded subtraction $\mathcal{A}$ such that the runs of $\mathcal{A}$ and $\mathcal{A}'$ are the same.*

Proof. Given $\mathcal{A}'$ construct $\mathcal{A}$ as follows: for every transition $t_{\mathcal{A}'} = (q, g, up, q')$ change the guard of this transition to $g' := g \wedge (\bigwedge_{x \in B} x \leq M_x)$ where B is the set of clocks x with update of the form $x := x - c$ with $c > 0$. From Definition 5.2 it then follows that, for every reachable configuration (q, v) in $\mathcal{A}$, if $v \models g$ then $v \models g'$. □

The $\mathcal{A}$-map computation with $\mathsf{pre}(\varphi, up)$ presented in Section 3.3.2 may not terminate for some timed automaton with syntactically bounded subtraction, however, due to the optimizations presented in Table 3.1 the reduced $\mathcal{A}$-map computation always terminates for this class of automata. This is proved in the following result. In the following, an $\mathcal{A}$-map (or a reduced $\mathcal{A}$-map) $\mathcal{G} \colon q \mapsto \mathcal{G}(q)$ is said to be *finite* if each of the sets $\mathcal{G}(q)$ is finite.

Lemma 5.6. *For every timed automaton with syntactically bounded subtraction $\mathcal{A}$, the reduced $\mathcal{A}$-map is finite, therefore, the reduced $\mathcal{A}$-map computation terminates for every timed automaton with syntactically bounded subtraction.*

Proof. Let $\mathcal{A}$ be a timed automaton with syntactically bounded subtraction. Let $\overline{M}$ be the maximum constant appearing among the guards and updates in $\mathcal{A}$. Define $\overline{\mathcal{G}}$ to be the set of *all* atomic constraints with constant at most $\overline{M}$. It will be shown that the map $\mathcal{G}$ assigning $\mathcal{G}(q) = \overline{\mathcal{G}}$ for all q, is a reduced $\mathcal{A}$-map. Since $\overline{\mathcal{G}}$ is a finite set, the map $\mathcal{G}$ is also finite, therefore the lemma will follow.

From the construction of $\overline{\mathcal{G}}$, it follows that the first two conditions of Definition 3.34 hold. It remains to be shown that $\mathsf{pre}(\varphi, g, up) \in \overline{\mathcal{G}}$, for every $\varphi \in \overline{\mathcal{G}}$ and for every transition (q, g, up, q') of $\mathcal{A}$. Choose a constraint $\varphi \in \overline{\mathcal{G}}$. Note that $\mathsf{pre}(\varphi, g, up)$ is a constraint having a larger constant than φ only if up contains subtractions (since the other possible update is only a reset to 0 in this class). Thus, if up does not contain subtractions, from the construction of $\overline{\mathcal{G}}$ it follows that $\mathsf{pre}(\varphi, g, up) \in \overline{\mathcal{G}}$. Now, if $up_x = x - c$ for some clock x and $c \in \mathbb{N}_{>0}$, then from Definition 5.4, the guard g contains $x \lhd_1 c_1$ for some $\lhd_1 \in \{<, \leq\}$ and $c_1 \in \mathbb{N}_{\geq 0}$. If $up^{-1}(\varphi)$ is some $x \lhd d$, then Case 1 of Table 3.1 gives $\mathsf{pre}(\varphi, g, up) = \top$. If $up^{-1}(\varphi)$ is $d \lhd x$, from Case 2 of the table, $\mathsf{pre}(\varphi, g, up) = c_1 \leq x$ or $\mathsf{pre}(\varphi, g, up) = d \lhd x$ with $d \leq c_1$, which are both present in $\overline{\mathcal{G}}$ by construction.

Finally, assume that $up^{-1}(\varphi)$ is a diagonal constraint $x - y \lhd d$ or $d \lhd x - y$ and Case 3 of Table 3.1 does not apply. Then, $up_x = x - c_1$ with $c_1 \geq 0$ and $up_y = y - c_2$ with $c_2 \geq 0$ (a reset for x or y is not possible). Moreover, if $c_1 > 0$ (resp. $c_2 > 0$) then g contains some $x \lhd_1 c_1'$ (resp. $y \lhd_2 c_2'$). If $c_1 > 0$ then, since Case 3 does not apply, $d \leq c_1' \leq \overline{M}$ and $up^{-1}(\varphi) \in \overline{\mathcal{G}}$. If $c_1 = 0$ and $c_2 > 0$ then the constraint φ is respectively $x - y \lhd d + c_2$ or $d + c_2 \lhd x - y$. Since $0 \leq d < d + c_2 \leq \overline{M}$, the constraint $up^{-1}(\varphi)$ is already in $\overline{\mathcal{G}}$. Therefore, the third condition of Definition 3.34 is also satisfied, proving that the map $\mathcal{G}$ is indeed a reduced $\mathcal{A}$-map. □

Given a timed automaton with bounded subtraction $\mathcal{A}$, an *equivalent* timed automaton with syntactically bounded subtraction $\mathcal{A}'$ can be constructed (thanks to Lemma 5.5). Due to Lemma 5.6, the reduced $\mathcal{A}'$-map will be finite. Therefore, the relation $\sqsubseteq_{\mathcal{G}}$ with such an $\mathcal{A}'$-map will yield a sound and complete procedure for checking reachability in the automaton $\mathcal{A}$. This gives an alternate proof of Theorem 5.3. This also makes a zone-based algorithm available for checking reachability unlike the region-based approach originally proposed in [FKPY07]. How schedulability can be modelled using timed automata with syntactically bounded subtraction will be discussed in Chapter 6.

5.3 Clock Bounded Reachability

In this thesis, so far, only the classical reachability problem has been discussed, that asks: given an updatable timed automaton, does there exist a run of the automaton terminating at a specified state? A modified reachability problem can also be asked: given an updatable timed automaton $\mathcal{A}$ and a bound $B \in \mathbb{R}_{\geq 0}$, does there exist a run $(q_0, v_0) \to (q_1, v_1) \to \cdots \to (q_n, v_n)$ of $\mathcal{A}$, ending at a specified state q_n,

satisfying the property that $v_i(x) \leq B$, for every clock x and every valuation v_i? This problem is called the *clock bounded reachability problem.*

This problem is decidable in general, since, there can only be finitely many zones reachable during the zone graph exploration. However, to optimize the enumeration through simulations, a simulation relation is required. If the $\mathcal{A}$-map can be computed then $\sqsubseteq_{\mathcal{G}}$ can be used for this purpose. It turns out, while considering clock bounded reachability, every input automaton can be modified (by adding constraints to its guards) into another automaton, preserving the clock-bounded behaviours and having finite reduced $\mathcal{A}$-map.

Given an updatable timed automaton $\mathcal{A}$, asking for clock bounded reachability is same as asking reachability in the automaton $\mathcal{A}_B$, where the automaton $\mathcal{A}_B$ is derived from $\mathcal{A}$ by adding the constraint $\bigwedge_{x \in X} x \leq B$ in every guard of $\mathcal{A}$. Clearly, whenever a state in $\mathcal{A}_B$ is reachable, in every accepting run the value of no clock can grow beyond the bound B. It turns out, for all such automata $\mathcal{A}_B$, the reduced $\mathcal{A}_B$-map computation terminates and therefore reachability can be checked in $\mathcal{A}_B$, using the reachability algorithm with the simulation $\sqsubseteq_{\mathcal{G}}$.

Theorem 5.7. *Given an updatable timed automaton $\mathcal{A}$ and a bound $B \in \mathbb{R}_{\geq 0}$, the reduced $\mathcal{A}_B$-map is finite.*

Proof. Let M be the maximum constant appearing in a guard of $\mathcal{A}_B$ (note that the constants in the updates are not considered). Let $\overline{\mathcal{G}}$ be the set of all atomic constraints with constant at most M, as defined in the proof of Lemma 5.6. It will be shown that $\mathsf{pre}(\varphi, g, up) \in \overline{\mathcal{G}}$, for every $\varphi \in \overline{\mathcal{G}}$ and for every transition (q, g, up, q') and thus the map $\mathcal{G} \colon q \mapsto \mathcal{G}(q)$ with $\mathcal{G}(q) = \overline{\mathcal{G}}$ is a reduced $\mathcal{A}_B$-map. The proof proceeds similarly to Lemma 5.6, although here there is an added convenience that the guard g contains an upper constraint for every clock. Suppose g contains $x \lhd c$. Then, when $up^{-1}(\varphi)$ is an upper constraint $x \lhd d$, $\mathsf{pre}(\varphi, g, up)$ is $\top$ (Case 1); when φ is $d \lhd x$, $\mathsf{pre}(\varphi, g, up)$ gives a constraint with constant atmost c; when φ is a diagonal constraint $x - y \lhd d$ or $d \lhd x - y$ and Case 3 does not apply then $d \leq c$ and hence $\mathsf{pre}(\varphi, g, up)$ is a constraint in $\overline{\mathcal{G}}$. □

Theorem 5.7 implies, given an updatable timed automaton $\mathcal{A}$ and a bound B, the clock bounded reachability problem can be checked by first constructing $\mathcal{A}_B$ and then using Algorithm 1 with the simulation relation $\sqsubseteq_{\mathcal{G}}$, where $\mathcal{G}$ is the reduced $\mathcal{A}_B$-map (can be computed using Algorithm 4).

5.4 Discussion

This chapter discussed the usefulness of the simulation relation $\sqsubseteq_{\mathcal{G}}$ defined in Chapter 3. It showed that this relation can be used for checking reachability in all the subclasses of UTA mentioned in [BDFP04], in particular, the full class of Timed Automata (containing diagonal constraints). However, non-deterministic updates discussed in [BDFP04] have not been considered. It was also shown that the relation $\sqsubseteq_{\mathcal{G}}$ can be used for checking reachability in Timed Automata with Bounded

Subtraction, therefore, while checking (preemptive) schedulability. Lastly, a modified version of reachability problem was discussed and shown that this relation $\sqsubseteq_{\mathcal{G}}$ can also be used for checking this problem.

Chapter 6

Implementation and experiments

This chapter provides a short note on the existing implementation of TChecker [HP] and the prototype implementation of the simulation relation $\sqsubseteq_G$ (Chapter 3) and the algorithm for checking the relation $(q, Z) \sqsubseteq_G (q, Z')$ (Chapter 4) in TChecker. This chapter further reports on the experiments performed using this prototype implementation. Two case studies are reported on two variants of the schedulability problem. The first case study (Section 6.2) is based on job-shop scheduling. In this problem, there are a set of tasks that need to be scheduled in a fixed number of reusable resources (machines). The model considered in this case study is a modified version of the timed automata modelling of job-shop scheduling described in [AAM06]. The original model considered only diagonal-free timed automata, the modified model is produced by adding diagonal constraints to the original model. This modified modelling is part of the work [GMS19]. The second case study (Section 6.3) considers a more general schedulability problem, where tasks gets *released* through a timed automaton – periodically or aperiodically. Also, while a task is being executed, another task is allowed to preempt this running task. The model considered in this case study is a modified version of the network of timed automata described in [FPY02] modelling preemptive job-shop scheduling. This modified modelling was presented in the work [GMS20].

The Timed Automata and Updatable Timed Automata models defined in Chapter 2 did not consider *events* (also known as *actions*) on transitions (Remark 1). Events are not relevant when considering the reachability problem in a single timed automaton. When reachability is checked in a network of timed automata, first, a *product* timed automaton is constructed from the network and then the reachability is checked on this product automaton. The events are important when constructing this product automaton. However, once the product automaton has been constructed, events become irrelevant for checking reachability. Since this thesis considers only the reachability problem – assuming, instead of a network of (updatable) timed automata, the product automaton is given – events on transitions have not been considered so far. However, while modelling real-time systems, events are indeed useful. When modelling a system as a network of (Updatable)

Timed Automata, the individual automata can *synchronize* with each other via events (also called *synchronizing actions*), present on the transitions. In a network, each individual automaton can have a set of events *local* to itself and also there can be certain events that are shared among multiple component automata of the network. These shared events are used as synchronizing events. For example, consider the two automata $\mathcal{A}_1$ and $\mathcal{A}_2$ in Figure 6.1. The event a is a synchronizing event in this network. The transition $q_0 \to q_1$ in $\mathcal{A}_1$, when taken, *emits* the signal a. Whereas, the transition $q'_0 \to q'_1$ in $\mathcal{A}_2$ *listens* to the signal a and this transition can only be taken when the signal a is emitted by some transition. Therefore, in this network, although the transition in $\mathcal{A}_2$ is enabled from the initial time itself (because of the guard $y \leq 20$, true when $y = 0$), it can only be taken once $y \geq 10$, because of the transition present in $\mathcal{A}_1$.

(a) The automaton $\mathcal{A}_1$ (b) The automaton $\mathcal{A}_2$

Figure 6.1: Network of two timed automata with a synchronizing event a

6.1 Implementation

This section gives a short description of the implementation of TChecker [HP] – an open source tool for checking reachability in diagonal-free Timed Automata. This, however, will not be an exhaustive description of the implementation of TChecker. This section describes the parts of TChecker that were required to be modified in order to implement the simulation relation $\sqsubseteq_{\mathcal{G}}$ (Chapter 3) and the algorithm for checking this relation between two zones (Chapter 4). A more detailed description of TChecker can be found in the wiki (https://github.com/ticktac-project/tchecker/wiki) and in the documentation built while installing TChecker.

The simulation relation $\sqsubseteq_{\mathcal{G}}$ has been implemented on TChecker v0.3 and can be found at https://github.com/mukherjee-sayan/tchecker.

6.1.1 A quick tour of TChecker

Parsing an input. TChecker can parse every updatable timed automaton (with updates only of the form $x := y + d$, for some clocks x, y and some integer d). This parsing happens in two steps. In the first step, an initial representation of the input (an object of type `system_declaration_t`) is built. In the next step, a more detailed model (an object of type `system_t`) is built. The zone graph is built from this final representation. This final model contains all the information about the states and transitions, with the invariants, guards and updates being parsed.

Building the zone graph. Reachability algorithms implemented in TChecker fix their own semantics of the zone graphs to be built. These get set through the function `factory` present in the namespace `zg`. The semantics set whether the

zones present in the zone graph are time-elapsed or not and also whether the zones are extrapolated or not. The folder `/src/tck-reach` contains implementations of the available algorithms. Each of the algorithms contain a `run` function, that first sets the semantics of the zone graph and then proceeds to check reachability. This `run` function also computes the appropriate clock bounds function required for the algorithm. This `run` function calls another `run` function (defined in the file `/include/tchecker/algorithms/covreach/algorithm.hh`). This second `run` function builds the reachability graph according to Algorithm 1. This reachability graph (an object of a particular `graph_t` class, defined in the reachability algorithm) gets built according to the semantics set for the zone graph.

Working with zones. Zones are implemented as Difference Bound Matrices (DBM) [Dil90] in TChecker. Each entry in a DBM is referred to as a difference bound (representing a pair of the form $(\triangleleft, c)$) and is implemented as the class `db_t`. All the operations on zones, including checking if one zone is simulated by (or contained in) the other, are defined in the file `/include/tchecker/dbm/dbm.hh`.

6.1.2 Constructing the parameters of $\sqsubseteq_G$

Implementation of the static analysis for computing the parameters of $\sqsubseteq_G$, consisted of two steps. First is a new data-structure to represent $\mathcal{A}$-map and second is the fixpoint computation (Algorithm 4) for computing the reduced $\mathcal{A}$-map (Definition 3.34). This new data structure is implemented as a class named `a_map_t` and it contains for each location (implemented as `loc_id_t` – an alias for `uint32_t`) of the automaton, two vectors `G` (containing only diagonal constraints), `Gdf` (containing only non-diagonal constraints). `a_map_t` is declared inside the namespace `amap` in the file `/include/tchecker/clockbounds/clockbounds.hh`.

The file `/include/tchecker/clockbounds/solver.hh` contains the definition of the fixpoint computation (Algorithm 4). The functions required for computing the reduced $\mathcal{A}$-map are kept inside the namespace `amap`. The function `compute_amap` computes the reduced $\mathcal{A}$-map given a system (an object of `system_t`). The fixpoint computation requires the computation of the constraint $\mathsf{pre}(\varphi, g, up)$ according to Table 3.1, this is computed by the function `compute_pre`. Given an input updatable timed automaton, whether the fixpoint computation will terminate or not, can be checked using the constant N described in Proposition 3.37. The function `find_cutoff_bound` computes this constant N. If a constraint with constant greater than N gets computed during the fixpoint computation, then an error is thrown stating that the tool is unable to check reachability for this input automaton, since the fixpoint computation will not terminate.

When reachability is checked for a network of Timed Automata, the clock bounds (L, U, M or $\mathcal{G}$) are first computed for the individual component automata. Then, a product automaton is constructed. Each state in this product automaton is a tuple of locations of the original network. For each such tuple $q_{prod} = (q_1, q_2, \ldots, q_n)$, its $\mathcal{G}(q_{prod})$ is defined as $\mathcal{G}(q_1) \cup \mathcal{G}(q_2) \cup \cdots \cup \mathcal{G}(q_n)$. This may yield wrong answers if the network involves clocks that are shared among more than one component automata. It is not yet clear how to compute $\mathcal{A}$-map in the presence of shared clocks. The

benchmarks that have been considered in this chapter do not involve shared clocks. The class `a_map_t` contains the function `bounds` that implements these sets $\mathcal{G}$ for tuples of locations (objects of the class `vloc_t`).

Remark 9. The input automata in TChecker can also contain invariants in states. When considering a state q of an input automaton, the atomic constraints present in the invariant of q are added to the set (implemented as a vector) $\mathcal{G}(q)$.

6.1.3 Implementing $Z \sqsubseteq_G Z'$

Given two zones Z and Z', the algorithm for checking $Z \sqsubseteq_G Z'$ (presented in Chapter 4) was required to be implemented in order to use the relation $\sqsubseteq_G$ in the reachability algorithm. Three functions have been implemented in this purpose, in the file `/src/dbm/dbm.cc`. The function `is_g_le` implements Algorithm 6 and the function `is_g_le_star` implements Algorithm 7. These functions take as arguments two dbms (representing two zones), a set of diagonal constraints and a set of non-diagonal constraints. Algorithm 5 has been implemented as the function `is_g_le_nd` for checking the relation $Z \sqsubseteq_{G^{nd}} Z'$, given two zones Z, Z' and a set of non-diagonal constraints G^{nd}.

TChecker also did not allow updates of the form $x := y - d$ where $d \in \mathbb{N}$. Although it could parse inputs with such updates, it could not check reachability in the presence of these updates. The function `reset` (also implemented in the file `/src/dbm/dbm.cc`) has been modified to incorporate updates with subtractions.

6.1.4 Running TChecker with $\sqsubseteq_{\mathcal{G}}$ simulation

While checking reachability in TChecker, some options need to be selected. First, the reachability algorithm needs to be chosen. This is done using the flag `-a`. A new algorithm option has been added named `gsim` that implements the reachability algorithm with the simulation relation described in this thesis. The label of the state for which reachability is being checked also needs to be provided while running TChecker, using the flag `-l`. Finally, the order in which the nodes of the zone graph are explored can be set while running TChecker, using the flag `-s` and selecting `bfs` or `dfs`. By default, `bfs` is used as the search order.

The file `/src/tck-reach/tck-reach.cc` contains the function `main` that sets these parameters provided while running TChecker on an input automaton. The new algorithm `gsim` sets the zone graph parameters as: (i) the zones present in the graph are time elapsed and (ii) no extrapolation used on zones. These parameters are set inside the `run` function present in the file `/src/tck-reach/zg-gsim.cc`.

The following command checks reachability with the simulation relation $\sqsubseteq_{\mathcal{G}}$:

```
tck-reach -a gsim -l <label-of-target-state> -s bfs
                    <path-to-input-file>
```

The next two sections describe two case studies, where the input models contain diagonal constraints and, in some examples, updates as well.

6.2 Job-Shop scheduling

The problem of job-shop scheduling has been modeled as (network of) timed automata by Abdeddaïm et al. in [AAM06]. Accepting runs of this model translates to feasible schedules. This job-shop scheduling problem considers the following scenario: there is a given set of *jobs*, each job J is an ordered sequence of *tasks* $\{(m_1, d_1), (m_2, d_2), \ldots, (m_k, d_k)\}$, where each pair (m_i, d_i) represents a task that needs to be executed by the resource m_i and it takes duration d_i. The set of all resources is finite. Each job also has a known deadline D. The problem is to know whether it is possible to execute all the jobs without missing any of the deadlines.

The model to be described in this section is an extension of the (diagonal-free) timed automata model introduced for job-shop scheduling. The benchmark models the following situation: there are n jobs $J_1, J_2, \ldots, J_n$ and k machines $m_1, m_2, \ldots, m_k$. Each job J_i consists of two tasks given by the tuples $T_i^1 := (m_i^1, a_i^1, b_i^1)$ and $T_i^2 = (m_i^2, a_i^2, b_i^2)$ where m, a, b (with appropriate indices) denote respectively the machine on which the task needs to be executed and $[a, b]$ is an interval such that the (not precisely known) time needed to execute the task falls in this interval. In addition to this, each job has an overall deadline D_i. The question now is: can these tasks be scheduled so that all jobs finish within their deadline? This problem has been modeled using diagonal-free Timed Automata. Now, add a further constraint (excluding the indices for clarity): for a job J with tasks T^1 and T^2, let t^1 and t^2 be the execution times. It is known that $t^1 \in [a^1, b^1]$ and $t^2 \in [a^2, b^2]$. Now, add a dependency between the tasks: if $t^1 \geq c^1$, then $t^2 \leq c^2$ and if $t^1 \leq d^1$ then $t^2 \geq d^2$ for suitable constants c^1, c^2, d^1, d^2. This says that, if the first task is executed for a longer time, the second task needs a shorter time to finish and vice versa. This kind of a dependency has a natural modeling using diagonal constraints. Figure 6.2 illustrates a part of the UPPAAL-style automaton for a job J capturing the timing requirements - this model makes use of the feature of committed locations in UPPAAL [BY03]. Time is not allowed to elapse in this state, and in a product automaton, the components with committed locations execute first. This modeling template of the job scheduling problem with dependent tasks can easily be extended when jobs have more tasks and there are similar dependencies between them. In addition to this automaton, the mutual exclusion between machines – each machine can have atmost one task at a time – is modeled by using additional boolean variables. The experiments are performed with varied number of jobs and this produces different timed automata with diagonal constraints. Note that, although this model is acyclic, there are "cross simulations" which help in pruning the search: the same state q can be reached by multiple paths and simulations help in cutting out new searches.

Each model reported in Table 6.1 is a product of k timed automata. In the table, the name of the model is written followed by the number k denoting the number of automata involved in the product. Along with this, the number of diagonal constraints present in each of them is also reported. The first three benchmarks make the algorithm enumerate the entire zone graph (by checking reachability of a non-existent state). Whereas, the last two benchmarks check if the jobs are schedulable (by checking if the state f of Figure 6.2 is reachable in all of the component

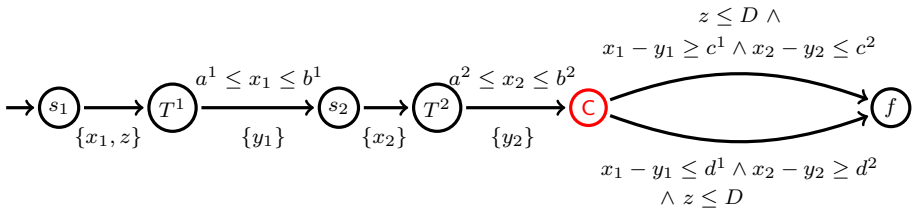


Figure 6.2: Timed automaton model capturing the timing requirements in a scheduling problem. The state C marked red is a *committed location*, which is a convenient modeling feature in UPPAAL and TChecker. Time is not allowed to elapse in such a state.

		TChecker with $\sqsubseteq_G$		UPPAAL	
Model	$\#\mathcal{D}$	Nodes count	time	Nodes count	time
Job Shop 3	12	206	0.008s	31711	23.318s
Job Shop 5	20	8459	1.795s	-	timeout
Job Shop 7	28	-	timeout	-	timeout
Job Shop sched 3	12	206	0.008s	31607	23.977s
Job Shop sched 4	20	1272	0.114s	-	timeout

Table 6.1: The column $\#\mathcal{D}$ gives the number of diagonal constraints. There are two methods reported in the table. First algorithm (TChecker with $\sqsubseteq_G$) is the implementation of simulation based reachability using state based guards and pre and the second algorithm (UPPAAL) is the result of running the model ($\mathcal{A}$) with diagonal constraints in UPPAAL. Nodes count denotes the number of nodes enumerated during a breadth-first exploration of the zone graph. The first three models compute the entire zone graph, whereas, the last two check if the jobs are schedulable. Time is reported in seconds and the timeout is set to 15 minutes.

automata). In the models reported, the values of the variables have been set to: $a^1 = 1,\ b^1 = 8,\ a^2 = 6,\ b^2 = 11,\ c^1 = 5,\ c^2 = 7,\ d^1 = 2,\ d^2 = 10$ and $D = 20$.

6.3 Preemptive scheduling

This section presents a model of a variant of the schedulability problem considered by Fersman et al. in [FPY02], with similar ideas but with simpler data structures. The problem considers a set of tasks and every task (t_i) has a fixed execution time (C_i) and a deadline (D_i). Once a task gets added to the task queue, it needs to be executed before its deadline. The scheduling policy allows *preemption*, that is, while a task is being executed, if a new task gets added to the queue, then the scheduler can preempt the current task, execute the new task and then resume executing the preempted task. The scheduler follows the Earliest Deadline First (EDF) scheduling strategy: the running task is the one with the closest deadline. There exists a queue containing tasks that need to be executed before each of their deadlines. The problem is to know whether it is possible to schedule all the

tasks under EDF, or whether atleast one of the tasks in the queue will violate its deadline. For simplicity in modelling, a restriction is put: at any point of time, there is only one instance of a particular task in the task queue. The network of timed automata in Figure 6.3 models this problem of schedulability. The network consists of the following automata: (i) a scheduler automaton – this chooses the task that needs to be run now, among the tasks present in the task queue, (ii) an automaton corresponding to each task – this automaton keeps track of the current status of the task, whether it is present in the queue, whether it is being executed at the moment, whether it has been preempted or whether it has missed its deadline, and (iii) a task release automaton – this timed automaton controls the release of tasks into the task queue, this can model periodic tasks or non-periodic tasks.

Scheduler automaton. For every task t_i, a boolean variable $queued_i$ is used to denote whether the task t_i has been added to the queue ($queued_i = 1$) or not ($queued_i = 0$). The *scheduler automaton* selects which task gets executed depending on the *time left to reach its deadline*. The automaton has a state to remember that the task queue is currently empty ($queue = \emptyset$). The state *taskrunning* denotes that one of the tasks (present in the queue) is being executed. The remaining states, all marked red, constitute the gadget that chooses the task to be executed. A state is marked red to denote that it is a *committed state*, meaning no time is allowed to elapse in this state. Assuming there are n tasks, there are n *layers* of these states. In the i^{th} layer, there are $(i+1)$-many states, $temp_{i0}, temp_{i1}, \ldots, temp_{ii}$. The set of all tasks is assumed to be ordered. The state $temp_{ij}$ in the automaton denotes that among the first i tasks, $t_1, t_2, \ldots, t_i$ (some of these tasks might not be queued), task t_j (which must be queued) has the closest deadline among the tasks that are queued. The state $temp_{i0}$ denotes that none of the first i tasks are queued. Note that, after checking the first i tasks, if the task with the earliest deadline is t_j, then after checking the $(i+1)^{th}$ task, the task with the earliest deadline can either be $t_{(i+1)}$ or remain t_j. The automaton thus has a transition from $temp_{ij}$ to $temp_{(i+1)(i+1)}$ checking if the deadline of t_{i+1} is (strictly) closer than the deadline of t_j. This is checked by the guard $D_{i+1} - d'_{i+1} < D_j - d'_j$, where d'_j is a clock that gets reset as soon as the task t_j gets added to the queue (i.e. on every transition with synchronization $release_j$). Otherwise, after checking the $(i+1)^{th}$ task, t_j remains to be the task with the earliest deadline. This is possible in two scenarios - (i) the task $t_{(i+1)}$ is not present in the queue ($queued_{i+1} = 0$) or (ii) the deadline of $t_{(i+1)}$ is atleast as far as the deadline of t_j (checked by the guard $D_j - d'_j \leq D_{i+1} - d'_{i+1}$). There are these two edges from $temp_{ij}$ to $temp_{(i+1)j}$, for every $i = 1, 2, \ldots, n-1$ and $1 \leq j \leq i$. The states $temp_{ni}$ (for $i = 1, 2, \ldots, n$) denote that among all the n tasks, the task t_i has the earliest deadline and hence this task must be executed. This is ensured using the transition $temp_{ni} \rightarrow taskrunning$ with the synchronization run_i. This "triangle-like" gadget selects the task with the earliest deadline among the tasks present in the queue. This gadget can be modified to model other scheduling strategies in place of EDF.

While a task is being executed, that is, while the automaton is at the state *taskrunning*, if a task gets released, then the scheduler automaton again needs to choose the task with the earliest deadline, among the queued tasks (including this

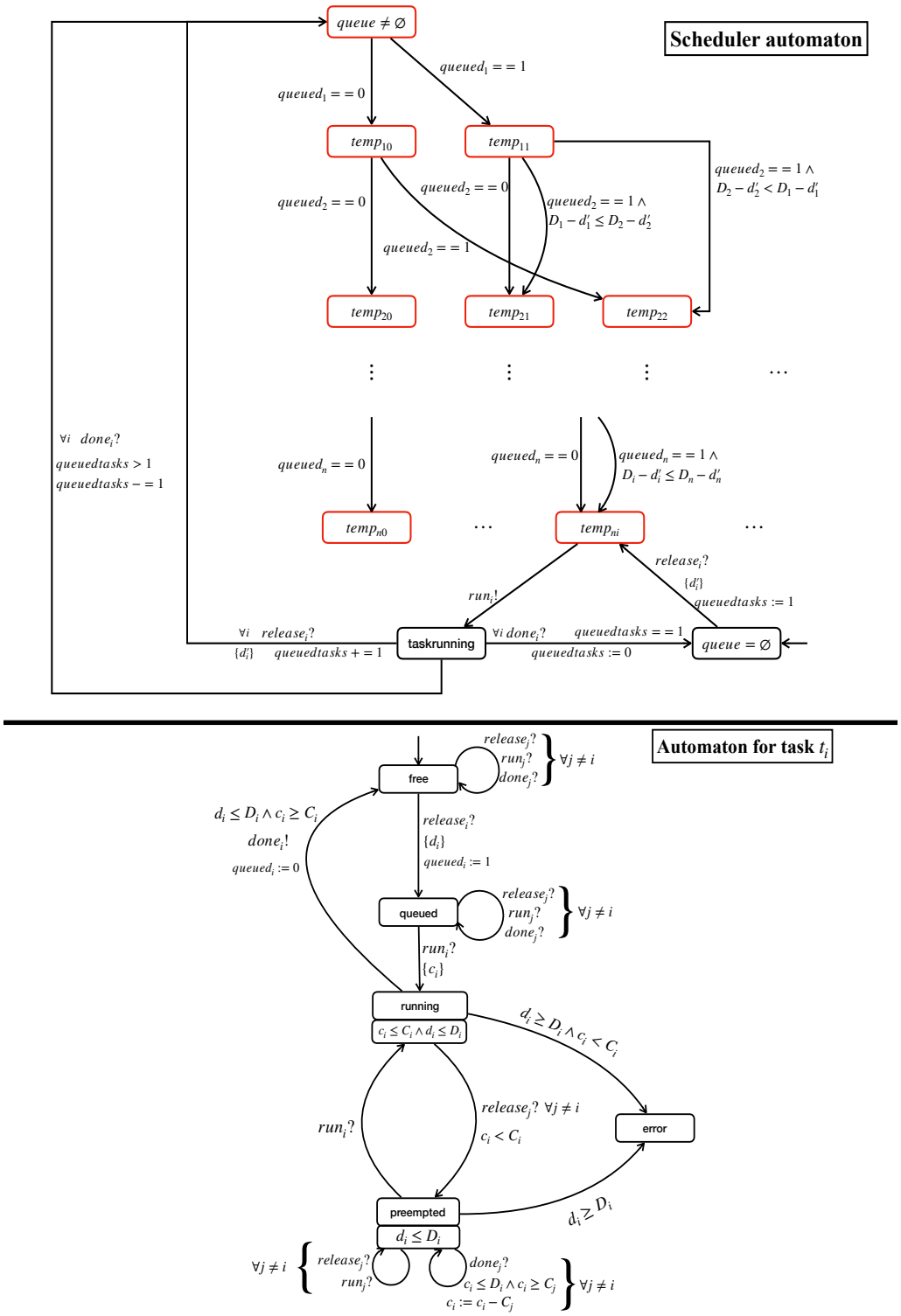


Figure 6.3: EDF scheduler and task handler

newly queued task). For this, there is an edge *taskrunning* → '*queue* ≠ ∅' with the synchronization $release_i$, for every $i = 1, 2, \ldots, n$.

After a task gets finished, there are two cases - either the queue becomes empty (the edge *taskrunning* → '*queue* = ∅') or the queue remains non-empty (the edge *taskrunning* → '*queue* ≠ ∅'). These edges are synchronized using the signal $done_i$ for every $i = 1, 2, \ldots, n$.

Automaton for individual tasks. The *automaton for task* t_i maintains the state of a task, whether the task has been added to the task queue (*queued*), whether it is running (*running*) or it has been preempted (*preempted*) or if the deadline of this task has been violated (*error*). The state $free$ denotes that the task has not been released yet. As soon as the task gets released, a clock d_i gets reset, that tracks the deadline of the task. This is done in the transition $free \to queued$. (This clock d_i is essentially a *copy* of the clock d'_i present in the scheduler automaton. These copies are used to avoid using shared clocks.) Another clock c_i is used to maintain the total execution time of the task. This clock gets reset as soon as the task starts to be executed, on the transition *queued* → *running*. Since a task can be released while another task is being executed, assuming there are n many tasks, there are $(n-1)$ many edges *running* → *preempted* on each of the signals $release_j$ where $j \neq i$. While a task is preempted, there can be other tasks getting released, run or finished. In order to account for the cases for releasing and executing of other tasks, there is a self loop on each of the signals $release_j$ and run_j, for $j \neq i$. Since no other instance of t_i gets released into the queue while the task t_i is preempted, there is no self loop on either $release_i$ or run_i. Note, while the automaton is in the state *preempted*, the clock c_i keeps elapsing time, although the task is not being executed. So, while the task t_i is preempted, when another task t_j gets finished (communicated through the synchronization $done_j$), the computation time of t_j is deducted from the clock c_i. This is done using the update $c_i := c_i - C_j$ on the edge *preempted* $\xrightarrow{done_j?}$ *preempted*. In this edge, there is also the guard $c_i \leq D_i$. This ensures that this automaton (for task t_i) is a timed automaton with bounded subtraction. Therefore, the *reduced* $\mathcal{A}$-map computation terminates for this automaton, as described in Chapter 5. In the state *running*, the invariant $c_i \leq C_i \wedge d_i \leq D_i$ ensures that the task is yet to finish its execution and also the deadline has not been violated. The invariant at the state *preempted* also ensures that, while at this state, the deadline of the task has not been violated yet. There are two edges to the *error* state, from *running* and from *preempted*, both checking that the deadline is going to be (or has been) violated. The edge *running* → $free$ ensures that the task has been executed within its deadline. The self loops on the states $free$ and *queued* ensure that there is no deadlock when another task gets released or being executed or has been done.

Task release automaton. The release of a task is controlled using a *task release automaton*. Two such automata have been considered for the experiments. These are depicted in Figure 6.4 and Figure 6.5. Both of these automata ensure that a task t_i gets released only if it is not already present in the task queue (i.e. $queued_i = 0$).

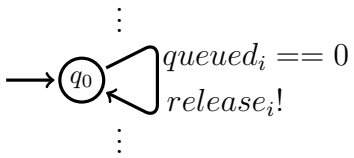


Figure 6.4: $\mathcal{A}_{flower}$: non-deterministically releases a task whenever the task is not already present in the queue

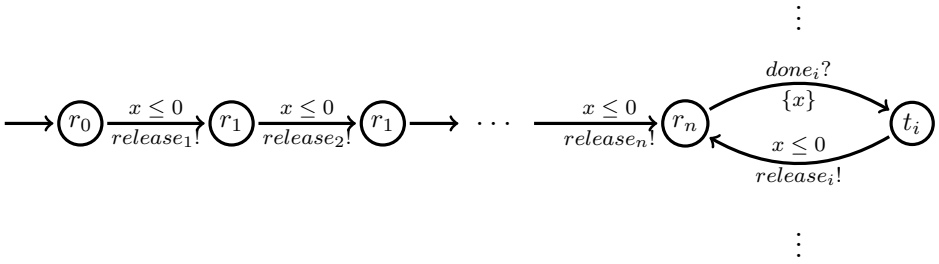


Figure 6.5: $\mathcal{A}_{wc}$: first release all the tasks in zero time, then release a task as soon as the queued instance finishes its execution

Periodic task release automaton. A *periodic task* with a period P is a task, that gets released after every P time units. In order to ensure the fact that not more than one instance of a task gets added to the task queue, the deadline for each periodic task considered for the experiments, is lesser than or equal to its period. A set of periodic tasks is modeled using the automaton in Figure 6.6. This automaton uses a clock p_i for each task t_i to compute the time till its last release. Now, all the tasks are first released before any time has elapsed (ensured by the guards $x \leq 0$). After this, whenever a task has not been released for its period, it gets released. This is ensured by the self loops in the state r_n.

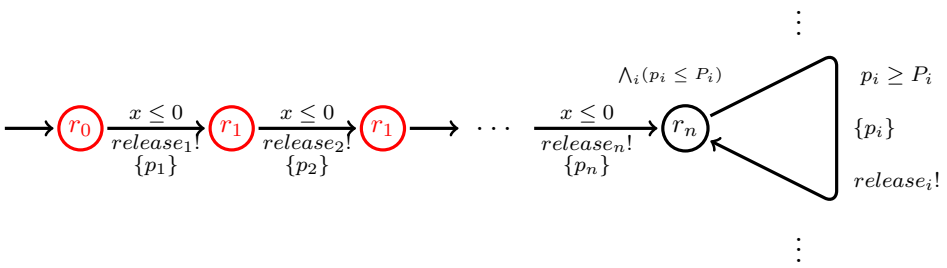


Figure 6.6: $\mathcal{A}_{periodic}$: task releasing automaton for a set of periodic tasks $\{t_i \mid i = 1, 2, \ldots, n\}$. The state r_n has an invariant $\bigwedge_i (p_i \leq P_i)$. The states $r_0, \ldots, r_{n-1}$ are marked red to denote that they are *committed states*. This automaton first releases all tasks in zero time, then releases the task t_i after every P_i time units, for $i = 1, 2, \ldots, n$, P_i being the period of task t_i.

Some preliminary experiments of preemptive scheduling are reported in Table 6.2. The *SporadicPeriodic* model is from the tool TIMES [AFM+03]. This example has three periodic tasks and one sporadic task, that is controlled by a task

releasing automaton. Multiple examples are run for different values of a constant N present in the task releasing automaton. For each of these models, the answer to schedulability matches with the answer of TIMES. The *Mine-Pump* example is presented in [GCO01]. This example has six periodic tasks. The benchmark considers only the first five tasks – $(58, 200, 200)$, $(37, 250, 250)$, $(37, 300, 300)$, $(39, 350, 350)$, $(33, 800, 800)$, $(33, 1000, 1000)$, where each task is represented as (*computation time*, *deadline*, *period*). For this benchmark as well, the answer to schedulability matches with the result in [GCO01].

Model	Schedulable?	Nodes count	time
SporadicPeriodic-5	Yes	677	0.028s
SporadicPeriodic-20	No	852	0.031s
Mine-Pump	Yes	31352	8.654s
Flower task triggering automaton: (computation time, deadline)			
(1,2), (1,2), (1,2)	No	204	0.006s
(1,10), (1,10), (1,10), (1,4)	Yes	87475	73.695s
Worst-case task triggering automaton: (computation time, deadline)			
(1,2), (1,2), (1,2)	No	20	0.001s
(1,10), (1,10), (1,10), (1,4)	Yes	441	0.026s

Table 6.2: Nodes Count is the number of nodes enumerated during a breadth-first-search; *Flower* task triggering automaton is given in Figure 6.4 and *Worst-case* task triggering automaton is given in Figure 6.5

6.4 Miscellaneous examples

This section reports on the performance on three miscellaneous examples in Table 6.3. The first example (Fischer) is based on the model provided in [Rey07]. The second model (Cex) is the automaton presented in [Bou03] with an added parameter to make the model larger as discussed in [Rey07]. The third example is the automaton given in Figure 3.3. The first two examples in this section contain diagonal constraints and only resets, whereas, the third example contains subtraction updates as well as diagonal constraints.

6.5 Discussion

For the benchmarks containing only resets (Table 6.1 and first two sets of examples in Table 6.3), two algorithms have been considered. For each algorithm, the number of nodes enumerated and the time taken are reported. The experiments were run on a laptop with 2.3 GHz Dual-core processor, 8 GB RAM, running macOS 11.

The first algorithm (TChecker with $\sqsubseteq_G$) gives a significant gain over the second algorithm (UPPAAL) both in the nodes count and time. A brief explanation of this

		TChecker with $\sqsubseteq_G$		UPPAAL	
Model	# $\mathcal{D}$	Nodes count	time	Nodes count	time
Fischer 3	3	104	0.002s	4272	0.129s
Fischer 4	4	458	0.015s	357687	321.110s
Fischer 5	5	1904	0.151s	-	timeout
Fischer 7	6	29187	12.282s	-	timeout
Cex 1	2	7	0.0001s	26	0.034s
Cex 2	4	141	0.005s	2180	0.037s
Cex 3	6	3109	0.553s	182394	119.980s
Cex 4	8	62762	51.493s	-	timeout
Figure 3.3 $\times$ 1	1	3	0.0001s	-	-
Figure 3.3 $\times$ 3	3	54	0.019s	-	-
Figure 3.3 $\times$ 5	5	978	18.664s	-	-

Table 6.3: Nodes count is the number of nodes enumerated during a breadth-first exploration of the zone graph; # $\mathcal{D}$ denotes the number of diagonal constraints present in the model

phenomenon is provided below. The performance of the reachability algorithm is dependent on three factors:

- the parameters on which the extrapolation or simulation is based on: the maximum constant based M-simulations which use the maximum constant appearing in the guards, versus the LU-simulations which make a distinction between lower bound guards $c \lhd x$ and upper bound guards $x \lhd c$ (refer to [BBLP06] for the exact definitions of extrapolations based on these parameters, and [HSW16] for simulations based on these parameters); LU-simulations are superior to M-simulations.

- the way the parameters are computed: global parameters which associate a bound to each clock versus the more local state based parameters which associate a set of bounds functions to each state [BBFL03]; local bounds are superior to global bounds.

- when diagonal constraints are present, whether zones get split or not: each time a zone gets split, new enumerations start from each of the new nodes; clearly, a no-splitting-of-zones approach is superior to zone splitting.

The algorithm implemented in TChecker uses the superior heuristic in all the three optimizations above. The no-splitting-of-zones was possible thanks to the simulation based approach, which temporarily splits zones for checking $Z \sqsubseteq_G Z'$ (Theorem 4.28), but never starts a new exploration from any of the split nodes. To the best of our knowledge, the algorithm for handling diagonal constraints implemented in UPPAAL 4.1 does not use the three optimizations. In particular, it is not clear how the extrapolation approach can avoid the zone splitting in an efficient manner. The improvement of the $\sqsubseteq_G$ based approach gets amplified when bigger products with many more diagonals are considered.

Along with the method of handling diagonal constraints in UPPAAL reported in this section, another method would be to compute the diagonal-free equivalent of the automaton and then using the diagonal-free engine present in UPPAAL. Both of these approaches are in principle prone to a $2^{\#\mathcal{D}}$ blowup compared to the first approach, where $\mathcal{D}$ gives the number of diagonal constraints. The table shows that a good extent of this blowup indeed happens. Comparisons with two other works dealing with the same problem have not been reported in the table: the refined diagonal free conversion [Rey07] and the LU simulation extended with LU^d simulation for diagonals discussed in the work [GMS18]. However, the results reported in this thesis are better than the tables reported in these papers.

Chapter 7

Conclusion

This chapter provides a brief summary of the results obtained in this thesis. It then concludes by listing down a few research directions to be explored in future.

This thesis considered the reachability problem in Updatable Timed Automata (UTA) and also, in particular, its subclass, Timed Automata (with or without diagonal constraints). The restricted class of diagonal-free Timed Automata enjoy an efficient zone based reachability algorithm. This thesis tried to adapt this algorithm to handle diagonal constraints and updates. The zone based reachability algorithm is not guaranteed to terminate, unless a suitable simulation relation is used. Therefore, in order to adapt the algorithm, a simulation relation was required to be defined and also in order to use this relation in the reachability algorithm, an algorithm had to be devised to check this relation between two zones.

This thesis proposed the relation $\sqsubseteq_G$ – parameterized by a set of atomic constraints (Definition 2.1). This relation was first defined between two valuations (Definition 3.1) and subsequently was lifted to a relation between two zones (Definition 3.4). The goal was to make this relation a simulation relation for UTA.

Before trying to make $\sqsubseteq_G$ a simulation relation, this relation $\sqsubseteq_G$ was first compared with the $\sqsubseteq_{LU}$ simulation relation available for diagonal-free Timed Automata. Given a set of non-diagonal constraints G, the L, U bounds can be defined corresponding to that G (Definition 3.12). It turned out, given such a G and its corresponding L, U, the relation $\sqsubseteq_G$ relates more valuations (Theorem 3.13) and therefore more zones (Corollary 3.14) than $\sqsubseteq_{LU}$.

Simulation relations (Definition 2.16) are, firstly, relations between two configurations (having the same state) of UTA. The relation $\sqsubseteq_G$ has, therefore, been further lifted to a relation $\sqsubseteq_{\mathcal{G}}$ between two configurations (Definition 3.15) and then to a relation between two nodes of the zone graph (Definition 3.16). While defining these two relations, instead of using a single G for every state of the automaton, a family of sets $\mathcal{G} = \{\mathcal{G}(q) \mid q \text{ is a state of the updatable timed automaton}\}$ was used. This parameter $\mathcal{G}$ had to be computed carefully to ensure $\sqsubseteq_{\mathcal{G}}$ becomes a simulation relation for UTA, and therefore also for Timed Automata.

Chapter 3 described a suitable parameter $\mathcal{G}$ (Definition 3.34) that made $\sqsubseteq_{\mathcal{G}}$ a simulation relation (Theorem 3.36). A fixpoint computation was proposed to compute this parameter (Lemma 3.35, Algorithm 3). This fixpoint computation, however, was not guaranteed to terminate for every updatable timed automata. A method for checking when this computation terminates was also presented. It was proved that during the fixpoint computation, if a constraint with "large enough" constant gets added to one of the sets $\mathcal{G}(q)$, then the fixpoint computation will not terminate (Proposition 3.37). Given an updatable timed automaton, deciding if the fixpoint computation terminates or not was shown to be in PTIME when the constants in the automaton are represented in unary, whereas, it is PSPACE-complete when the constants are encoded in binary (Theorem 3.43). This termination check, although hard, does not add an overhead to the reachability procedure, since it is done on-the-fly, while computing the parameter $\mathcal{G}$. The non-termination of the fixpoint computation is not a problem, since Chapter 5 proved that the fixpoint computation indeed terminates for the known decidable subclasses of UTA.

Chapter 4 then proved that the relation $\sqsubseteq_G$ is, in fact, finite (Theorem 4.1). This means whenever $\sqsubseteq_G$ can be used (with suitable G) as the simulation relation, the reachability algorithm will terminate. Since $\sqsubseteq_G$ can be used as soon as the fixpoint computation terminates and since checking reachability is undecidable in UTA, the fixpoint computation cannot terminate for every updatable timed automaton.

After proving this finiteness of $\sqsubseteq_G$, Chapter 4 dealt with the second goal of the thesis – an algorithm for checking the relation $\sqsubseteq_G$ between two zones. Devising this algorithm made the relation $\sqsubseteq_G$ usable in a reachability algorithm.

To begin with, Section 4.2 gave an algorithm (Algorithm 5) for checking the relation $Z \sqsubseteq_G Z'$, when G does not contain diagonal constraints. In this case, it was shown that if $Z \not\sqsubseteq_G Z'$, then it is due to at most two constraints present in G (Theorem 4.8). This gave a quadratic time algorithm for checking this relation, that matched the bound for checking the $\sqsubseteq_{LU}$ simulation between zones.

When G is also allowed to contain diagonal constraints, it turned out, checking $Z \sqsubseteq_G Z'$ can be reduced to checking two $\sqsubseteq_{G^-}$ relations, where G^- consists of all the constraints of G minus one diagonal constraint (Theorem 4.28). This result translated into a recursive algorithm (Algorithms 6 and 7) for checking $Z \sqsubseteq_G Z'$. This algorithm used the algorithm developed for the diagonal-free case, at most exponentially (in the number of diagonal constraints present in G) many times. However, the algorithm contains some heuristics that can help decide the relation with fewer diagonal-free checks. Chapter 4 concluded by showing that checking $Z \not\sqsubseteq_G Z'$, when G contains diagonal constraints is, in fact, NP-complete (Theorem 4.30). The hardness was proved by showing a reduction from 3-SAT (Lemma 4.38).

Finally, Chapter 6 reported on the implementations of the relation $\sqsubseteq_G$, the fixpoint computation (Algorithm 4) and the algorithm for checking $\sqsubseteq_G$ between two zones, in the tool TChecker. Some preliminary experiments were reported based on two variants of the schedulability problem. Section 6.2 reported on the classic job-shop scheduling problem. This problem has already been modelled using Timed Automata, a slightly modified version of the problem was considered in this section that allows the use of diagonal constraints while modelling. The more

complex preemptive scheduling problem was discussed in Section 6.3. This problem has been modelled using Timed Automata with Bounded Subtraction [FKPY07], a subclass of Updatable Timed Automata, containing diagonal constraints. An alternate modelling (to make it implementable in the version of TChecker that has been used in the experiments) of this problem was provided and the experimental results were reported. These results show improvement over the existing methods.

Future directions

Following are a few questions that can be explored in future:

- While considering Updatable Timed Automata, this thesis has not considered the non-deterministic updates $x :\sim y + d$ where $\sim \in \{<, \leq, \neq, \geq, >\}$. How to construct the parameter so that $\sqsubseteq_G$ remains a simulation relation even in the presence of these kind of updates?

- When the parameter of the proposed simulation relation contains smaller component sets, it results in more simulations and therefore smaller zone graphs. The improvement in [GMS20] from [GMS19] suggests that the proposed static analysis (for computing the parameter) may not be the optimum one. The question that therefore remains is: given an updatable timed automaton, is it possible to compute an even better parameter that results in more number of simulations?

- With a zone based algorithm now available for Timed Automata with diagonal constraints and with some of the updates, can the various optimizations studied for diagonal-free Timed Automata – for example, to avoid the state-space explosion problem, the *local time* semantics based technique studied by Govind et al. in [GHSW19] – be lifted to these classes as well?

- Some extensions of Timed Automata have been studied – for example, *Pushdown Timed Automata* studied by Akshay et al. in [AGP21] – that use zone based algorithm together with an off-the-shelf simulation relation. The simulation relation proposed in this thesis can also be plugged into such algorithms. Moreover, since this relation can also handle diagonal constraints and updates, can those extended models be further extended with these features?

- Along with Timed Automata, another tool that is used for modelling real-time systems is Timed Logic. The satisfiability of formulae in timed logics can sometimes be checked by first converting the formula into a timed automaton and then checking emptiness of the language of that automaton. Can diagonal constraints (or updates) be used to get 'smaller' automata while converting a logic formula, which may then possibly speed up the satisfiability check?

- Event Clock Automata (ECA) defined by Alur et al. in [AFH99] is a model that, unlike Timed Automata, enjoys determinizability. This model uses *event clocks* instead of the usual clocks used in Timed Automata. The notions of

zones have been defined for ECA by Geeraerts et al. in [GRS11]. They also show that when the zone graph of timed automata is adapted to these kinds of zones, its computation is still not guaranteed to terminate without a *widening* operator. Can we define our simulation relation in terms of event clocks to get a suitable widening operator for Event Clock Automata?

- Parametric Timed Automata (PTA) defined by Alur et al. in [AHV93] is also an extension of Timed Automata that has been used in practice. In this class, the guards present in the transitions of the automaton contain parameters in place of constants. Although most of the decision problems are undecidable in this class, several syntactic subclasses have been considered and several decision problems have also been studied – [And19] contains a survey. Can the simulation relation proposed in this thesis be used in the context of PTAs?

Bibliography

[AAM06] Yasmina Abdeddaïm, Eugene Asarin, and Oded Maler. Scheduling with timed automata. *Theor. Comput. Sci.*, 354(2):272–300, 2006. doi:10.1016/j.tcs.2005.11.018.

[AAS12] Parosh Aziz Abdulla, Mohamed Faouzi Atig, and Jari Stenman. Dense-timed pushdown automata. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*, pages 35–44. IEEE Computer Society, 2012. doi:10.1109/LICS.2012.15.

[AD90] Rajeev Alur and David Dill. Automata for modeling real-time systems. In Michael S. Paterson, editor, *Automata, Languages and Programming (ICALP)*, pages 322–335, Berlin, Heidelberg, 1990. Springer Berlin Heidelberg. doi:10.1007/BFb0032042.

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, April 1994. doi:10.1016/0304-3975(94)90010-8.

[AFH99] Rajeev Alur, Limor Fix, and Thomas A. Henzinger. Event-clock automata: A determinizable class of timed automata. *Theor. Comput. Sci.*, 211(1-2):253–273, 1999. doi:10.1016/S0304-3975(97)00173-4.

[AFM+03] Tobias Amnell, Elena Fersman, Leonid Mokrushin, Paul Pettersson, and Wang Yi. TIMES: A tool for schedulability analysis and code generation of real-time systems. In *FORMATS*, volume 2791 of *Lecture Notes in Computer Science*, pages 60–72. Springer, 2003.

[AGP21] S. Akshay, Paul Gastin, and Karthik R. Prakash. Fast zone-based algorithms for reachability in pushdown timed automata. In Alexandra Silva and K. Rustan M. Leino, editors, *Computer Aided Verification - 33rd International Conference, CAV 2021, Virtual Event, July 20-23, 2021, Proceedings, Part I*, volume 12759 of *Lecture Notes in Computer Science*, pages 619–642. Springer, 2021. doi:10.1007/978-3-030-81685-8_30.

[AHV93] Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. Parametric real-time reasoning. In S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal, editors, *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 592–601. ACM, 1993. doi:10.1145/167088.167242.

[AM04] Rajeev Alur and P. Madhusudan. Decision problems for timed automata: A survey. In *SFM*, volume 3185 of *Lecture Notes in Computer Science*, pages 1–24. Springer, 2004. doi:10.1007/978-3-540-30080-9_1.

[And19] Étienne André. What's decidable about parametric timed automata? *Int. J. Softw. Tools Technol. Transf.*, 21(2):203–219, 2019. doi:10.1007/s10009-017-0467-0.

[And21] Étienne André. IMITATOR 3: Synthesis of timing parameters beyond decidability. In Alexandra Silva and K. Rustan M. Leino, editors, *Computer Aided Verification - 33rd International Conference, CAV 2021, Virtual Event, July 20-23, 2021, Proceedings, Part I*, volume 12759 of *Lecture Notes in Computer Science*, pages 552–565. Springer, 2021. doi:10.1007/978-3-030-81685-8_26.

[ATP01] Rajeev Alur, Salvatore La Torre, and George J. Pappas. Optimal paths in weighted timed automata. In Maria Domenica Di Benedetto and Alberto L. Sangiovanni-Vincentelli, editors, *Hybrid Systems: Computation and Control, 4th International Workshop, HSCC 2001, Rome, Italy, March 28-30, 2001, Proceedings*, volume 2034 of *Lecture Notes in Computer Science*, pages 49–62. Springer, 2001. doi:10.1007/3-540-45351-2_8.

[BBFL03] Gerd Behrmann, Patricia Bouyer, Emmanuel Fleury, and Kim Guldstrand Larsen. Static guard analysis in timed automata verification. In *Tools and Algorithms for the Construction and Analysis of Systems, 9th International Conference (TACAS)*, pages 254–277, 2003. doi:10.1007/3-540-36577-X_18.

[BBLP06] Gerd Behrmann, Patricia Bouyer, Kim Guldstrand Larsen, and Radek Pelánek. Lower and upper bounds in zone-based abstractions of timed automata. *Int. J. Softw. Tools Technol. Transf.*, 8(3):204–215, 2006. doi:10.1007/s10009-005-0190-0.

[BC05] Patricia Bouyer and Fabrice Chevalier. On conciseness of extensions of timed automata. *J. Autom. Lang. Comb.*, 10(4):393–405, April 2005. doi:10.25596/jalc-2005-393.

[BCM16] Patricia Bouyer, Maximilien Colange, and Nicolas Markey. Symbolic optimal reachability in weighted timed automata. In Swarat Chaudhuri and Azadeh Farzan, editors, *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada,*

July 17-23, 2016, Proceedings, Part I, volume 9779 of *Lecture Notes in Computer Science*, pages 513–530. Springer, 2016. doi:10.1007/978-3-319-41528-4_28.

[BD00] Béatrice Bérard and Catherine Dufourd. Timed automata and additive clock constraints. *Inf. Process. Lett.*, 75(1-2):1–7, 2000. doi:10.1016/S0020-0190(00)00075-2.

[BDFP00a] Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury, and Antoine Petit. Are timed automata updatable? In E. Allen Emerson and A. Prasad Sistla, editors, *Computer Aided Verification, 12th International Conference (CAV)*, volume 1855 of *Lecture Notes in Computer Science*, pages 464–479. Springer, 2000. doi:10.1007/10722167_35.

[BDFP00b] Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury, and Antoine Petit. Expressiveness of updatable timed automata. In *MFCS*, volume 1893 of *Lecture Notes in Computer Science*, pages 232–242. Springer, 2000. doi:10.1007/3-540-44612-5_19.

[BDFP04] Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury, and Antoine Petit. Updatable timed automata. *Theor. Comput. Sci.*, 321(2-3):291–345, 2004. doi:10.1016/j.tcs.2004.04.003.

[BDGP98] Béatrice Bérard, Volker Diekert, Paul Gastin, and Antoine Petit. Characterization of the expressive power of silent transitions in timed automata. *Fundam. Inf.*, 36(2,3):145–182, August 1998. doi:10.3233/FI-1998-36233.

[BER94] Ahmed Bouajjani, Rachid Echahed, and Riadh Robbana. On the automatic verification of systems with continuous variables and unbounded discrete data structures. In Panos J. Antsaklis, Wolf Kohn, Anil Nerode, and Shankar Sastry, editors, *Hybrid Systems II, Proceedings of the Third International Workshop on Hybrid Systems, Ithaca, NY, USA, October 1994*, volume 999 of *Lecture Notes in Computer Science*, pages 64–85. Springer, 1994. doi:10.1007/3-540-60472-3_4.

[BFH+01] Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim Guldstrand Larsen, Paul Pettersson, Judi Romijn, and Frits W. Vaandrager. Minimum-cost reachability for priced timed automata. In Maria Domenica Di Benedetto and Alberto L. Sangiovanni-Vincentelli, editors, *Hybrid Systems: Computation and Control, 4th International Workshop, HSCC 2001, Rome, Italy, March 28-30, 2001, Proceedings*, volume 2034 of *Lecture Notes in Computer Science*, pages 147–161. Springer, 2001. doi:10.1007/3-540-45351-2_15.

[BJLY98] Johan Bengtsson, Bengt Jonsson, Johan Lilius, and Wang Yi. Partial order reductions for timed systems. In Davide Sangiorgi and Robert de Simone, editors, *CONCUR '98: Concurrency Theory, 9th International Conference, Nice, France, September 8-11, 1998, Proceedings*,

volume 1466 of *Lecture Notes in Computer Science*, pages 485–500. Springer, 1998. doi:10.1007/BFb0055643.

[BLR05] Patricia Bouyer, François Laroussinie, and Pierre-Alain Reynier. Diagonal constraints in timed automata: Forward analysis of timed systems. In Paul Pettersson and Wang Yi, editors, *Formal Modeling and Analysis of Timed Systems (FORMATS)*, pages 112–126, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. doi:10.1007/11603009_10.

[BMS13] Patricia Bouyer, Nicolas Markey, and Ocan Sankur. Robustness in timed automata. In Parosh Aziz Abdulla and Igor Potapov, editors, *Reachability Problems - 7th International Workshop, RP 2013, Uppsala, Sweden, September 24-26, 2013 Proceedings*, volume 8169 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2013. doi:10.1007/978-3-642-41036-9_1.

[Bou03] Patricia Bouyer. Untameable timed automata! In Helmut Alt and Michel Habib, editors, *STACS 2003*, pages 620–631, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. doi:10.1007/3-540-36494-3_54.

[Bou04] Patricia Bouyer. Forward analysis of updatable timed automata. *Formal Methods Syst. Des.*, 24(3):281–320, 2004. doi:10.1023/B:FORM.0000026093.21513.31.

[BY03] Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In *Lectures on Concurrency and Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 87–124. Springer, 2003.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, April 1986. doi:10.1145/5397.5399.

[CJ99] Hubert Comon and Yan Jurski. Timed automata and the theory of real numbers. In Jos C. M. Baeten and Sjouke Mauw, editors, *CONCUR '99: Concurrency Theory, 10th International Conference, Eindhoven, The Netherlands, August 24-27, 1999, Proceedings*, volume 1664 of *Lecture Notes in Computer Science*, pages 242–257. Springer, 1999. doi:10.1007/3-540-48320-9_18.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[Dil90] David L. Dill. Timing assumptions and verification of finite-state concurrent systems. In Joseph Sifakis, editor, *Automatic Verification Methods for Finite State Systems (CAV)*, pages 197–212, Berlin, Heidelberg, 1990. Springer Berlin Heidelberg. doi:10.1007/3-540-52148-8_17.

[DT98] Conrado Daws and Stavros Tripakis. Model checking of real-time reachability properties using abstractions. In Bernhard Steffen, editor, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 313–329, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg. doi:10.1007/BFb0054180.

[DY96] C. Daws and S. Yovine. Reducing the number of clock variables of timed automata. In *17th IEEE Real-Time Systems Symposium (RTSS)*, pages 73–81, Dec 1996. doi:10.1109/REAL.1996.563702.

[FJ15] John Fearnley and Marcin Jurdzinski. Reachability in two-clock timed automata is pspace-complete. *Inf. Comput.*, 243:26–36, 2015. URL: https://doi.org/10.1016/j.ic.2014.12.004, doi:10.1016/j.ic.2014.12.004.

[FKPY07] Elena Fersman, Pavel Krcál, Paul Pettersson, and Wang Yi. Task automata: Schedulability, decidability and undecidability. *Inf. Comput.*, 205(8):1149–1172, 2007.

[FPY02] Elena Fersman, Paul Pettersson, and Wang Yi. Timed automata with asynchronous processes: Schedulability and decidability. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 2280 of *Lecture Notes in Computer Science*, pages 67–82. Springer, 2002.

[FQSW20] Martin Fränzle, Karin Quaas, Mahsa Shirmohammadi, and James Worrell. Effective definability of the reachability relation in timed automata. *Inf. Process. Lett.*, 153, 2020. doi:10.1016/j.ipl.2019.105871.

[GCO01] Thorsten Gerdsmeier and Rachel Cardell-Oliver. Analysis of scheduling behaviour using generic timed automata. *Electronic Notes in Theoretical Computer Science*, 42:143 – 157, 2001. Computing: The Australasian Theory Symposium (CATS 2001).

[GHSW19] R. Govind, Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. Revisiting local time semantics for networks of timed automata. In Wan J. Fokkink and Rob van Glabbeek, editors, *30th International Conference on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands*, volume 140 of *LIPIcs*, pages 16:1–16:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.CONCUR.2019.16.

[GMS18] Paul Gastin, Sayan Mukherjee, and B. Srivathsan. Reachability in timed automata with diagonal constraints. In Sven Schewe and Lijun Zhang, editors, *29th International Conference on Concurrency Theory, CONCUR 2018, September 4-7, 2018, Beijing, China*, volume 118 of *LIPIcs*, pages 28:1–28:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.CONCUR.2018.28.

[GMS19] Paul Gastin, Sayan Mukherjee, and B. Srivathsan. Fast algorithms for handling diagonal constraints in timed automata. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I*, volume 11561 of *Lecture Notes in Computer Science*, pages 41–59. Springer, 2019. doi:10.1007/978-3-030-25540-4_3.

[GMS20] Paul Gastin, Sayan Mukherjee, and B Srivathsan. Reachability for Updatable Timed Automata Made Faster and More Effective. In Nitin Saxena and Sunil Simon, editors, *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020)*, volume 182 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 47:1–47:17, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.FSTTCS.2020.47.

[GRS11] Gilles Geeraerts, Jean-François Raskin, and Nathalie Sznajder. Event clock automata: From theory to practice. In Uli Fahrenberg and Stavros Tripakis, editors, *Formal Modeling and Analysis of Timed Systems - 9th International Conference, FORMATS 2011, Aalborg, Denmark, September 21-23, 2011. Proceedings*, volume 6919 of *Lecture Notes in Computer Science*, pages 209–224. Springer, 2011. doi:10.1007/978-3-642-24310-3_15.

[HHW97] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. HYTECH: A model checker for hybrid systems. *Int. J. Softw. Tools Technol. Transf.*, 1(1-2):110–122, 1997. doi:10.1007/s100090050008.

[HKPV98] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? *J. Comput. Syst. Sci.*, 57(1):94–124, 1998. doi:10.1006/jcss.1998.1581.

[HOW16] Christoph Haase, Joël Ouaknine, and James Worrell. Relating reachability problems in timed and counter automata. *Fundam. Informaticae*, 143(3-4):317–338, 2016. URL: https://doi.org/10.3233/FI-2016-1316, doi:10.3233/FI-2016-1316.

[HP] Frédéric Herbreteau and Gerald Point. Tchecker. URL: https://github.com/ticktac-project/tchecker.

[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 Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen, editors, *36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2016, December 13-15, 2016, Chennai, India*, volume 65 of *LIPIcs*, pages 48:1–48:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.FSTTCS.2016.48.

[HSW16] Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. Better abstractions for timed automata. *Inf. Comput.*, 251:67–90, 2016. doi:10.1016/j.ic.2016.07.004.

[KLM+15] Gijs Kant, Alfons W. Laarman, Jeroen Meijer, Jaco van de Pol, Stefan Blom, and Tom van Dijk. Ltsmin: High-performance language-independent model checking. In *TACAS*, 2015.

[KNSS00] Marta Z. Kwiatkowska, Gethin Norman, Roberto Segala, and Jeremy Sproston. Verifying quantitative properties of continuous probabilistic timed automata. In Catuscia Palamidessi, editor, *CONCUR 2000 - Concurrency Theory, 11th International Conference, University Park, PA, USA, August 22-25, 2000, Proceedings*, volume 1877 of *Lecture Notes in Computer Science*, pages 123–137. Springer, 2000. doi:10.1007/3-540-44618-4_11.

[LPY97] Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.

[LRST09] Didier Lime, Olivier H. Roux, Charlotte Seidner, and Louis-Marie Traonouez. Romeo: A parametric model-checker for Petri nets with stopwatches. In Stefan Kowalewski and Anna Philippou, editors, *15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2009)*, volume 5505 of *Lecture Notes in Computer Science*, pages 54–57, York, United Kingdom, March 2009. Springer.

[NMA+02] Peter Niebert, Moez Mahfoudh, Eugene Asarin, Marius Bozga, Oded Maler, and Navendu Jain. Verification of timed automata via satisfiability checking. In Werner Damm and Ernst-Rüdiger Olderog, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems, 7th International Symposium, FTRTFT 2002, Co-sponsored by IFIP WG 2.2, Oldenburg, Germany, September 9-12, 2002, Proceedings*, volume 2469 of *Lecture Notes in Computer Science*, pages 225–244. Springer, 2002. doi:10.1007/3-540-45739-9_15.

[Rey07] Pierre-Alain Reynier. Diagonal constraints handled efficiently in UPPAAL. In *Research report LSV-07-02*, Laboratoire Spécification et Vérification. ENS Cachan, France, 2007.

[RSM19] Victor Roussanaly, Ocan Sankur, and Nicolas Markey. Abstraction refinement algorithms for timed automata. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I*, volume 11561 of *Lecture Notes in Computer Science*, pages 22–40. Springer, 2019. doi:10.1007/978-3-030-25540-4_2.

[SLDP09] Jun Sun, Yang Liu, Jin Song Dong, and Jun Pang. Pat: Towards flexible verification under fairness. In Ahmed Bouajjani and Oded Maler, editors, *Computer Aided Verification*, pages 709–714, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[THV+17] Tamás Tóth, Ákos Hajdu, András Vörös, Zoltán Micskei, and István Majzik. Theta: A framework for abstraction refinement-based model checking. In *2017 Formal Methods in Computer Aided Design (FMCAD)*, pages 176–179, 2017. doi:10.23919/FMCAD.2017.8102257.

[TYB05] Stavros Tripakis, Sergio Yovine, and Ahmed Bouajjani. Checking timed büchi automata emptiness efficiently. *Formal Methods Syst. Des.*, 26(3):267–292, 2005. doi:10.1007/s10703-005-1632-8.

[Wan04] Farn Wang. Efficient verification of timed automata with bdd-like data structures. *Int. J. Softw. Tools Technol. Transf.*, 6(1):77–97, 2004. doi:10.1007/s10009-003-0135-4.

[Yov97] Sergio Yovine. Kronos: A verification tool for real-time systems. *International Journal on Software Tools for Technology Transfer*, 1(1-2):123–133, 1997.