N° d'ordre :

THÈSE

PRÉSENTÉE À

L'UNIVERSITÉ BORDEAUX 1

ÉCOLE DOCTORALE DE MATHEMATIQUES ET D'INFORMATIQUE

Par **Balaguru SRIVATHSAN**

POUR OBTENIR LE GRADE DE

DOCTEUR

SPÉCIALITÉ : **INFORMATIQUE**

Abstractions pour les automates temporisés

Soutenue le: 6 Juin 2012

Après avis des rapporteurs:

Patricia Bouyer ····· (DR) *LSV, CNRS & ENS Cachan*
Joost-Pieter Katoen ····· (PR) *RWTH-Aachen*

Devant la commission d'examen composée de:

Ahmed Bouajjani	(PR)	*LIAFA, Univ. Paris Diderot (Paris 7)*	Examinateur
Patricia Bouyer	(DR)	*LSV, CNRS & ENS Cachan*	Rapporteuse
Bruno Courcelle	(PR)	*LaBRI, Université Bordeaux 1*	Président
Frédéric Herbreteau	(MCF)	*LaBRI, Université Bordeaux 1*	Directeur
Joost-Pieter Katoen	(PR)	*RWTH-Aachen*	Rapporteur
Igor Walukiewicz	(DR)	*LaBRI, CNRS & Université Bordeaux 1*	Directeur
James Worrell	(PR)	*Oxford university*	Examinateur

- 2012 -

Acknowledgements

First and foremost, I would like to thank my supervisors Igor and Frédéric. They have been a constant source of support all along the last three years.

To start with Igor, I admire the way he approaches a problem, his initial pre-processing by removing all the "garbage". This has been of crucial help in the problem solving involved in the thesis. I have also learnt from Igor the quality of aesthetics in research - making proofs elegant, choosing appropriate notations, creating accessible papers and presentations. I carry an utmost respect for Igor and am proud to be his student.

With Frédéric, I have had plenty of long discussions to clearly understand different concepts. He has been patient and willing to devote a good bit of his time. He has offered a tremendous amount of effort in building the tool incorporating our results and performing experiments. In the final days before submitting the thesis, he took pains to go through the entire thesis a couple of times, to carefully pick out errors. It was heartening to see his belongingness towards my thesis and it has motivated me to deliver better.

Another person directly involved in the making of this thesis is Dileep, whose internship here at LaBRI laid the seeds for the work in the first part of the thesis. I thank him for all the intense discussions we used to have on this topic. I indeed learnt a lot during his internship.

I was introduced to automata theory by Prof Krishna at IIT Bombay. Her lively lectures have generated all my interest in automata theory. She is also the one who introduced me to timed automata. She has been genuinely supportive in all my research endeavours and I owe a lot of gratitude to her.

My friends here in Bordeaux have given me a beautiful life outside my work. I will cherish the memorable times spent in their company. I would like to also thank my friends from IIT Bombay who have been an integral part of my life.

I owe my entire interest in mathematics to Prof Balakrishnan, my mathematics tutor for the IIT-JEE, the entrance examination to get into the IITs. The four years that I spent under his tutorship have paved the way for my career. I use the skills that I learnt from him till date. His approach to solving problems by visualizing them using diagrams has helped me a lot in tackling most of the problems solved in this thesis. No words would be sufficient to express how grateful I am to Prof Balakrishnan.

Last, I thank my parents for the indomitable cooperation they have shown towards my academics. In spite of a lot of hardships, they have never compromised on my education. This thesis is dedicated to my parents.

Résumé

Cette thèse revisite les problèmes d'accessibilité et de vivacité pour les automates temporisés.

L'accessibilité est couramment résolue par le calcul d'un arbre de recherche abstrait. L'abstraction est paramétrée par des bornes provenant des gardes de l'automate. Nous montrons que l'abstraction $\mathfrak{a}_{\preccurlyeq LU}$ de Behrmann *et al.* est la plus grande abstraction saine et complète pour les bornes LU. N'étant pas convexe, elle n'est pas mise en œuvre dans les outils. Nous introduisons une méthode qui permet son utilisation efficace. Finalement, nous proposons une optimisation des bornes à la volée exploitant le calcul de l'arbre.

Le problème de vivacité requiert de détecter les exécutions Zenon/non-Zenon. Une solution standard ajoute une horloge à l'automate. Nous montrons qu'elle conduit à une explosion combinatoire. Nous proposons une solution qui évite ce problème pour une grande classe d'abstractions. Pour les abstractions LU nous montrons que détecter ces exécutions est un problème NP-complet.

Mots clés: Automates temporisés, abstractions, algorithmes à la volée, exécutions Zenon

Abstract

We consider the classic model of timed automata introduced by Alur and Dill. Two fundamental properties one would like to check in this model are *reachability* and *liveness*. This thesis revisits these classical problems.

The reachability problem for timed automata asks if there exists a run of the automaton from the initial state to a given final state. The standard solution to this problem constructs a search tree whose nodes are abstractions of zones. For effectiveness, abstractions are parameterized by maximal lower and upper bounds (LU-bounds) occurring in the guards of the automaton. Such abstractions are also termed as LU-abstractions. The $\mathfrak{a}_{\preccurlyeq LU}$ abstraction defined by Behrmann *et al* is the coarsest known LU-abstraction. Although it is potentially most productive to use the $\mathfrak{a}_{\preccurlyeq LU}$ abstraction, it has not been used in implementations as it could lead to non-convex sets. We show how one could use the $\mathfrak{a}_{\preccurlyeq LU}$ abstraction efficiently in implementations. Moreover, we prove that $\mathfrak{a}_{\preccurlyeq LU}$ abstraction is optimal: given only the LU-bound information, it is the coarsest possible abstraction that is sound and complete for reachability. We then concentrate on ways to get better LU-bounds. In the standard procedure the LU-bounds are obtained from a static analysis of the automaton. We propose a new method to obtain better LU-bounds on-the-fly during exploration of the zone graph. The potential gains of proposed improvements are validated by experimental results on some standard verification case studies.

The liveness problem deals with infinite executions of timed automata. An infinite execution is said to be Zeno if it spans only a finite amount of time. Such runs are considered unrealistic. While considering infinite executions, one has to eliminate Zeno runs or dually, find runs that are non-Zeno. The Büchi non-emptiness problem for timed automata asks if there exists a non-Zeno run visiting an accepting state infinitely often. The standard solution to this problem adds an extra clock to take care of non-Zenoness. We show that this solution might lead to an exponential blowup in the search space. We propose a method avoiding this blowup for a wide class of abstractions weaker than LU-abstractions. We show that such a method does not exist for LU-abstractions unless P=NP. Another question related to infinite executions of timed automata is to decide the existence of Zeno runs. We provide the first complete solution to this problem. It works for a wide class of abstractions weaker than LU. Yet again, we show the solution could lead to a blowup for LU-abstractions, unless P=NP.

Keywords: Timed automata, finite abstractions, on-the-fly algorithms, Zenoness

Contents

Chapter 1

Introduction

Embedded computer systems are ubiquitous. Be it a vending machine serving beverages or a weather satellite keeping us well informed about the everyday weather, a number of devices carry an embedded computer system that receives certain inputs and executes specified actions. Additionally, these embedded systems might be required to perform the actions within precise time deadlines. For instance, a pacemaker used to regulate the heart beat should respond to a missed beat within the normal beat-to-beat period, or a car brake system should activate the necessary hardware to stop the car within a fraction of a second. Such time-critical systems subject to real-time constraints are commonly known as real-time systems. As is evident, it is of utmost importance to produce reliable real-time systems and therefore it is indispensable to have a framework *verifying* that the system meets the required specifications.

Timed automata [AD94] provide a convenient formalism to model and verify real-time systems. This formalism has been implemented in a number of tools [BDL+06, BDM+98, Wan04] and has been used in many different industrial case studies [HSLL97, BBP04]. A timed automaton is a finite automaton augmented with a set of *clocks*. The clocks start with a value zero and increase at the same rate along with time. Clocks can be used to constrain transitions: every transition is supplemented with a conjunction of inequalities, each of which compares a clock to a natural number. This conjunction of inequalities present in a transition is called the *guard* of the transition. The transition can be taken only at an instant when the values of the clocks satisfy the guard associated to the transition. To get something more interesting out of this finite structure, one allows for the power to *reset* values of clocks back to zero after taking a transition. The reset operation allows to measure time intervals between events. As is the case with finite automata, there are initial and final states in a timed automaton too. An example of a timed automaton is given in Figure 1.1.

Timed automata are the basis of many more sophisticated models. They

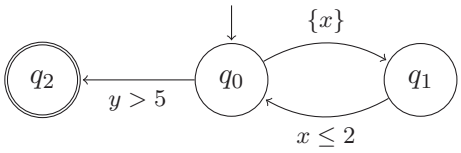


Figure 1.1: A timed automaton $\mathcal{A}_1$

can be extended with richer reset operations to give updatable timed automata [BDFP04, Bou04], with probabilistic edges to give probabilistic timed automata [GJ95, KNSS02, KNP11] or with costs for staying in a state to give priced timed-automata [BFLM11]. As these extended models are strongly based on timed automata, advances in this model could highly benefit the extensions. This brings us to study the basic timed automata of [AD94] in this thesis.

Two fundamental properties in verification are *reachability* and *liveness*. Reachability refers to asking if there exists an execution of the automaton from an initial state to a final state. This is useful in checking if some bad state of a system could possibly be reached. Liveness asks if there exists an infinite time-diverging execution of the automaton visiting an accepting state infinitely often. This problem is interesting since timed automata model reactive systems that continuously interact with the environment. Essentially, liveness asks if eventually, a good event keeps recurring forever and is different in spirit to reachability. Many common properties can be verified using either a reachability query or a liveness query. These two problems are known to be decidable for timed automata [AD94]. This thesis revisits these classical problems.

1.1 Reachability

The reachability problem, as mentioned above, asks if there exists an execution of the automaton from an initial state to a given final state. Reachability problem has been shown to be decidable in the paper introducing timed automata [AD94]. This is the fundamental property that makes the model amenable for verification. The basic reachability algorithm has undergone a lot of improvements over the years and is now the core of many real-time verification tools [BDL$^+$06, BDM$^+$98, Wan04].

Consider Figure 1.2. It shows the different behaviours of the automaton of Figure 1.1. Starting from the initial state q_0 and the initial clock valuation $\langle 0, 0\rangle$, the automaton can choose to spend an arbitrary amount of time at state q_0. When it takes the transition to q_1, the value of x is set to 0. So, if 1.3 time units are spent at q_0, the clocks take values $x = 1.3, y = 1.3$. But after taking the transition to q_1 in which x is reset, the values become

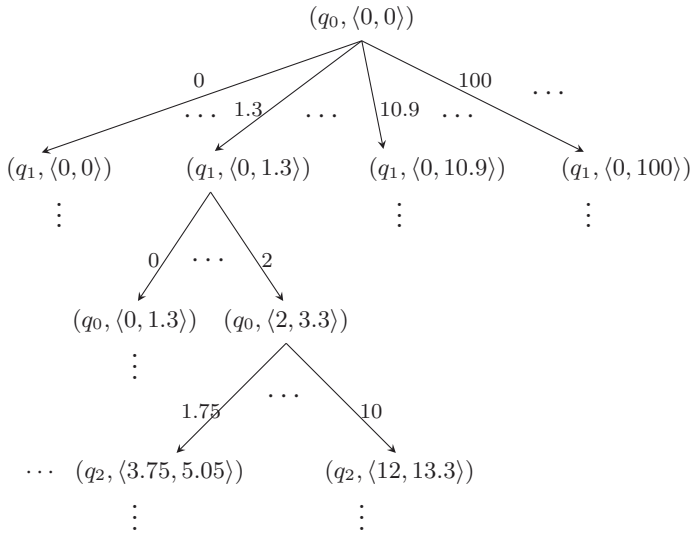


Figure 1.2: Part of the transition system showing the behaviours of the timed automaton $\mathcal{A}_1$ of Figure 1.1

$x = 0, y = 1.3$.

As we see, the space of configurations is uncountably infinite due to the dense time component. To solve the reachability problem, one needs to know if there is a sequence of enabled transitions leading to the final state. To know if a transition is enabled, one needs to check if the values of clocks before taking the transition satisfy the corresponding guard. This calls for an effective and efficient handling of the uncountably infinite space of clock valuations. This is the main challenge faced in the analysis.

The first solution to the reachability problem goes by a partition of the space of clock valuations into a finite number of *regions* [AD94]. Two valuations within a region are indistinguishable with respect to state reachability. The definition of regions is parameterized by a bound function M that associates to every clock x the maximum constant appearing in a guard involving x. For instance, the maximum bounds function M_1 for the automaton $\mathcal{A}_1$ of Figure 1.1 assigns $M_1(x) = 2, M_1(y) = 5$. The maximum bounds function also ensures finiteness of the number of regions. Once the space of valuations is partitioned into this finite number of regions, a cross product with the states of the automaton is taken to give what is called the *region graph*. It has been proved that the region graph is sound and complete for state reachability [AD94]. While this gives an effective solution to the reachability problem for timed automata, the number of regions is generally very big and hence the solution is impractical.

A more efficient solution uses sets of valuations called *zones*. Zones are convex sets described by constraints involving only the difference between

clocks. Due to this simple structure, zones can be efficiently represented using Difference Bound Matrices (DBMs) [Dil89]. This data structure can be efficiently manipulated and hence the algorithms using DBMs can be implemented efficiently [BY04]. There are two basic approaches to solve the reachability problem using zones: *forward analysis* and *backward analysis* [Bou09].

The forward analysis approach using zones explores the paths of the automaton starting from the initial state and collects the sets of valuations reachable at each step. It can be shown that these sets can be represented using zones [BY04]. The graph carved out this way is called the *zone graph* of the automaton. There is an execution of the automaton reaching an accepting state if and only if there is a node with an accepting state in the zone graph [DT98]. If the exploration of this zone graph reaches the final state, the algorithm halts reporting success. This way, the algorithm is on-the-fly as the entire automaton need not be explored. However, the zone graph explored naively in this forward analysis might potentially be infinite and the forward analysis algorithm might not terminate. The standard way out of this problem is to consider *abstractions of zones* instead of zones themselves, and to get a finite graph capturing the behaviour of the potentially infinite zone graph.

The backward analysis on the other hand starts from the final state along with the entire set of valuations. The algorithm explores the automaton backwards by doing a pre-computation of each transition, yielding the set of valuations that can pass through the respective transition. If the initial state of the automaton can be reached via this backward computation, the algorithm reports success. Contrary to the forward analysis approach, the backward analysis always terminates. However, it has been empirically observed that the backward analysis is computationally more expensive as compared to the forward analysis approach. In other words, the number of zones computed by backward analysis substantially exceeds the number obtained by forward analysis. Therefore, tools using zones and DBMs stick to the forward analysis approach [Bou09, BDL$^+$06, BDM$^+$98]. We consider the forward analysis approach in the thesis.

As seen above, abstractions of zones are crucial for termination of the forward reachability analysis. Figure 1.3 illustrates the idea. The picture on the left shows the zone graph obtained by forward analysis. Each node consists of a state of the automaton and a zone consisting of the set of valuations. Arrows indicate the edges (the guards and resets have not been explicitly shown). The target of the transition shows the state assumed by the automaton and the set of valuations obtained after taking the transition. Transitions marked with a cross are disabled as no valuation in the zone can satisfy the corresponding guard. The picture on the right shows an abstraction of the zone graph using an abstraction operator $\mathfrak{a}$ that is *sound*.

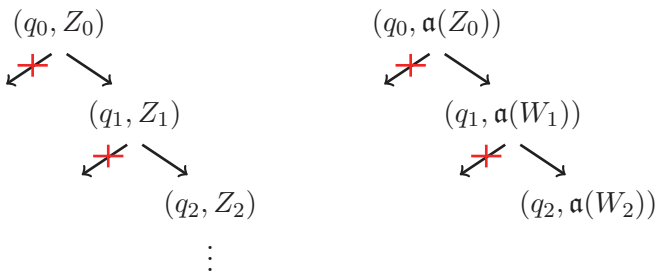


Figure 1.3: Abstracting the zone graph. Cross shows disabled edges.

We explain intuitively what it is to be sound. Consider the initial node $(q_0, \mathfrak{a}(Z_0))$ of the abstract graph. Transitions disabled from (q_0, Z_0) should be disabled from $(q_0, \mathfrak{a}(Z_0))$ too, as shown in the figure. Now the enabled transition is taken from $(q_0, \mathfrak{a}(Z_0))$. Since $\mathfrak{a}(Z_0)$ is a bigger set, we would get as a successor some (q_1, W_1) such that $Z_1 \subseteq W_1$. Before taking the transition from this new node, the algorithm abstracts W_1 to $\mathfrak{a}(W_1)$. The abstracted successor $(q_1, \mathfrak{a}(W_1))$ should now be sound with respect to (q_1, Z_1), as shown in the figure.

The goal is to come up with a sound abstraction operator that gives a finite abstract graph. *The coarser the abstraction, the smaller is the abstract graph obtained.* Therefore the aim is to get an abstraction operator as coarse as possible, but maintaining the criterion of soundness. To come up with an abstraction operator, a standard method is to use a *simulation* relation: a valuation v is simulated by a valuation v' if all paths taken by v can be taken by v' and for each enabled transition from v, the successor of v is simulated by the successor of v'. An abstraction based on a simulation relation adds to a set W the set of valuations that can be simulated by some valuation in W. By definition, the simulation relation entails soundness. Since we normally consider abstractions based on simulation relations, the abstract graph described above and illustrated on the right in Figure 1.3 is called a *simulation graph.*

It has been observed that computing the coarsest simulation relation given a timed automaton is EXPTIME-hard [LS00]. As reachability can be solved in PSPACE, this suggests that it may not be reasonable to solve reachability using the abstraction based on the coarsest simulation relation. We can get simulation relations that are computationally easier if we consider only a part of the structure of the automaton. The common way is to look at constants appearing in the guards of the automaton and consider them as parameters for abstraction.

There are two ways to obtain parameters. One way is to consider the same maximum bounds M as explained for the region graph. For every clock

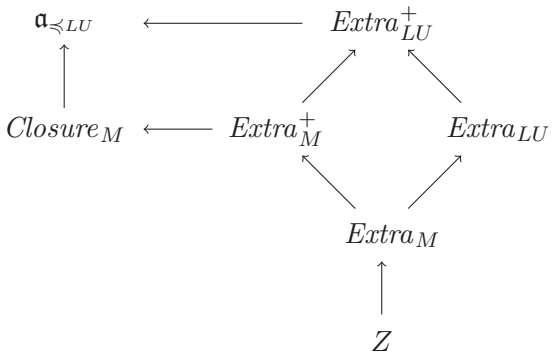


Figure 1.4: Hierarchy of abstractions in literature [BBLP06]. The ones starting with *Extra* are convex. $Extra^+_{LU}$ is currently used in implementations.

x, the maximum bounds function M associates the maximum constant appearing in a guard involving x. A more refined approach is to distinguish lower-bounding guards of the form $x > c$ or $x \geq c$ from upper-bounding guards of the form $x < c$ or $x \leq c$, giving two bound functions L and U [BBLP06]. The bound function L associates to every clock x the maximum constant appearing in a lower-bounding guard. Similarly U associates the maximum constant appearing in an upper-bounding guard. Abstractions using the LU-bounds are called LU-abstractions and those using the M-bounds are called M-abstractions. The LU-abstractions are generally coarser than the M-abstractions.

Figure 1.4 shows the sound and complete abstractions existing in the literature. The abstractions with subscript M are the M-abstractions and those with LU are the LU-abstractions. The figure shows the hierarchy of abstractions with respect to inclusion. The $\mathfrak{a}_{\preccurlyeq LU}$ abstraction is the coarsest known LU-abstraction subsuming the $Closure_M$ abstraction, which is the coarsest known among the M-abstractions. Therefore, it is potentially most productive to use the $\mathfrak{a}_{\preccurlyeq LU}$ abstraction in implementations. However, there is a catch. The $\mathfrak{a}_{\preccurlyeq LU}$ abstraction can lead to non-convex sets, in other words, sets that are not zones. Zones are objects that we know to efficiently represent and manipulate. On the other hand, we do not know of efficient ways to represent non-convex sets. Therefore, one adds the requirement that an abstraction operator yield a set that is a zone again. This would ensure that the simulation graph using this abstraction can be efficiently manipulated. Abstractions which map a zone to a zone again are called *convex* abstractions, as a zone is a convex set. In Figure 1.4 the abstraction operators in the right hand side diamond beginning with *Extra* are convex. Among the convex abstractions, since $Extra^+_{LU}$ is the coarsest, it is the one used in state-of-the-art implementations [BBLP06].

An allied line of research has been to look at ways to get better param-

What we already know	What we show in this thesis
Only convex abstractions in implementations [DT98, Bou04, BBLP06]	Efficient use of non convex abstractions $\mathfrak{a}_{\preccurlyeq_{LU}}$ and $Closure_M$ $\mathfrak{a}_{\preccurlyeq_{LU}}$ is optimal with respect to LU-bounds $Closure_M$ is optimal with respect to M-bounds
Bounds for abstraction by static analysis [BBFL03]	A new algorithm to get bounds on-the-fly during exploration

Table 1.1: Contributions towards the reachability problem

eters, that is better M-bounds and LU-bounds. The smaller the bounds, the coarser is the abstraction. Hence it is important to get as small bounds as possible, bearing in mind the constraint of soundness. The crude way is to look at the entire automaton and search for the relevant maximum constant. An improved approach associates a bound function to every state of the automaton [BBFL03]. In this approach, a static analysis is done on the automaton to search for guards *relevant* at a particular state. A guard involving clock x is relevant at a state q only if there is a path in the automaton starting from q and leading to the guard such that x is reset nowhere in the path. It has been observed that optimizing bounds using this static analysis reduces the size of the simulation graph substantially. The static analysis has therefore been adopted as the state-of-the-art approach to obtain bounds for abstraction.

Contributions of the thesis

We concentrate on two aspects of the reachability problem: efficient use of abstractions of zones, and calculating bounds for abstractions. We have highlighted the contributions of the thesis in Table 1.1. We explain below in more detail.

We give an efficient method to use non-convex abstractions $\mathfrak{a}_{\preccurlyeq_{LU}}$ and $Closure_M$ for reachability testing. As these abstractions are coarser than their convex counterparts, the simulation graphs obtained using these non-convex abstractions are *a priori* smaller. As a surprising bonus, we prove that $\mathfrak{a}_{\preccurlyeq_{LU}}$ is optimal: it is the coarsest abstraction depending only on LU-bounds that is sound and complete for reachability. Similarly, $Closure_M$ is

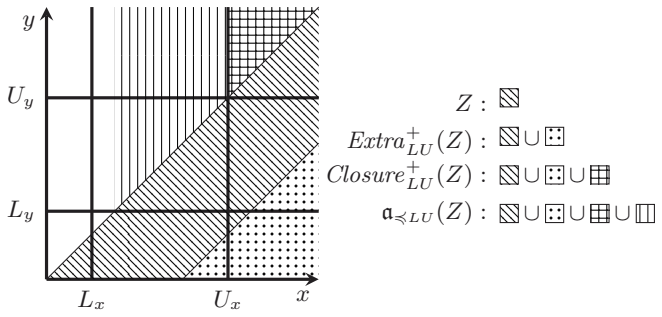


Figure 1.5: A comparison of abstraction operators for zones.

the coarsest abstraction depending on M-bounds that is sound and complete for reachability.

Since we know that $\mathfrak{a}_{\preccurlyeq LU}$ and $Closure_M$ are optimal respectively when LU-bounds and M-bounds are used, we cannot hope to look for better abstractions. However, we can strive to get better bounds themselves. The smaller the bounds, the coarser is the abstraction. As we have described above, the current method to obtain bounds performs a static analysis to assign bounds to each state of the automaton. We provide an algorithm that uses information gathered during the reachability analysis and assigns bounds on-the-fly to each node of the graph computed by the algorithm. An immediate gain is that bounds in the parts of the automaton that are unreachable become irrelevant. We have implemented our algorithm in a prototype tool and have checked the results on standard benchmarks. We report on some promising experimental results.

Related work

Forward analysis is the main approach for the reachability testing of real-time systems. The use of zone-based abstractions for termination has been introduced in [DT98]. In recent years, coarser sound abstractions have been introduced to improve efficiency of the analysis [BBLP06]. An approximation method based on LU-bounds, called $Extra^+_{LU}$, is used in the current implementation of UPPAAL [BDL$^+$06]. In [HKSW11] it has been shown that it is possible to efficiently use the region closure of $Extra^+_{LU}$, denoted $Closure^+_{LU}$. This has been the first efficient use of a non-convex approximation. A comparison of these abstractions is depicted in Fig. 1.5.

We have mentioned that abstractions are not needed in backward exploration of timed systems. Nevertheless, any feasible backward analysis approach needs to simplify constraints. For example [MPS11] does not use approximations and relies on an SMT solver instead. Clearly this approach is very difficult to compare with the forward analysis approach we study here.

Another related approach to verification of timed automata is to build a quotient graph of the semantic graph of the automaton with respect to some bisimulation relation [ACH+92, YL93, TY01, CHK08]. For reachability properties, this approach is not a priori competitive with respect to forward exploration as it requires to construct the whole state space of the automaton. It is more adapted to checking branching time properties.

1.2 Liveness

Timed automata model reactive systems that continuously interact with the environment. These reactive systems are also assumed to function forever. Therefore, it is interesting to consider infinite runs of a timed automaton. The liveness problem intuitively asks if there is a behaviour of the automaton where a good event keeps happening repeatedly.

In the case of infinite executions one has to eliminate the so called Zeno runs. These are executions that contain infinitely many steps taken in a finite time interval. For obvious reasons such executions are considered unrealistic. One way to treat Zeno runs would be to say that a timed automaton admitting such a run is faulty and should be disregarded. This gives rise to the problem of detecting the existence of Zeno runs in an automaton [BG06, GB07]. We call it the *Zenoness problem*. The other approach to handling Zeno behaviours is to say that due to imprecisions introduced by the modeling process one may need to work with automata having Zeno runs. This leads to the problem: given a timed automaton decide if it has a non-Zeno run passing through accepting states infinitely often. We call this the *Büchi non-emptiness* problem. This thesis gives new solutions to both the problems: the Büchi emptiness problem, and the Zenoness problem.

Büchi non-emptiness problem

There are two facets to the Büchi non-emptiness problem. Given a timed automaton, we first need to decide if it has an infinite run visiting accepting states infinitely often. Let us call this repeated state reachability. In addition to this, we need to ensure that the infinite run is non-Zeno.

Let us first consider repeated state reachability. It can be shown that the region graph is *sound and complete for repeated state reachability* [AD94]: an infinite path in the region graph can be instantiated to an infinite run of the timed automaton and vice versa. Due to the large size of the region graph, it is very difficult to use it efficiently in practice. Thanks to [TYB05, Tri99] we know that simulation graphs using M-abstractions are sound and complete for repeated state reachability. More recently in [Li09], it was shown that all simulation graphs using a time-abstract simulation [TAKB96] are sound and complete for reachability. This result entails that even LU-abstractions

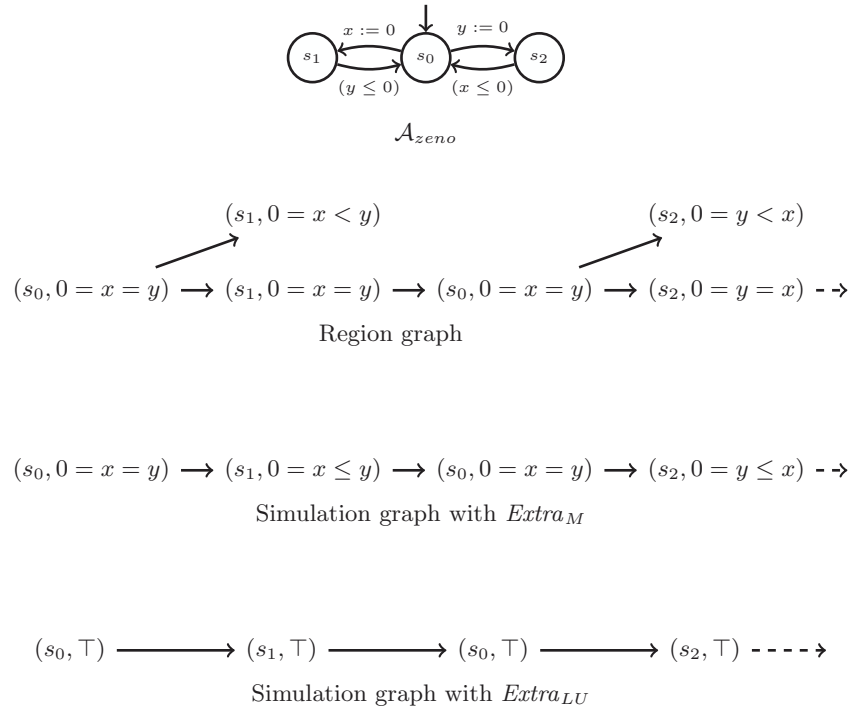


Figure 1.6: Non-Zeno runs from abstract paths

are good for repeated state reachability. However, it does not guarantee that the infinite run of the automaton that is obtained is non-Zeno.

Figure 1.6 shows an automaton $\mathcal{A}_{zeno}$ which has all runs Zeno. Paths of the region graph and simulation graphs using $Extra_M$ and $Extra_{LU}$ are depicted below in the same figure. In the region graph, the only infinite path corresponds to the run of the timed automaton taken without any time elapse. When time is elapsed at state s_0, the run cannot be continued further from state s_1. This is exactly incorporated in the region graph. However, in the simulation graphs, both the runs of the timed automaton are merged to a single path of the simulation graph. Moreover, the coarser the abstraction used, the lesser is the information in the simulation graph that one could tap to detect non-Zenoness.

Non-Zenoness can be detected directly from the region graph by checking for a *progress criterion* that can be encoded as an extra Büchi condition [AD94]. This progress criterion is however not sound for simulation graphs due to their coarseness as compared to regions. There exists a simple solution to this problem that amounts to transforming automata in such a way that every run passing through an accepting state infinitely often is non-Zeno. An automaton with such a property is called *strongly non-Zeno* [TYB05, Tri99]. The transformation is easy to describe and requires

What we already know	What we show in this thesis
SNZ construction for non-Zenoness [TYB05, Li09]	SNZ construction can cause exponential blowup of simulation graph A new construction which is polynomial for M-abstractions Deciding non-Zenoness from LU-abstract simulation graphs is NP-complete
Sufficient-only condition on automaton syntax for Zenoness [GB07]	A complete algorithm for Zenoness Complexity linear in size of simulation graph for M-abstractions Deciding Zenoness from LU-abstract simulation graphs is NP-complete

Table 1.2: Contributions towards the liveness problem (SNZ refers to Strongly non-Zeno).

addition of one new clock. Tools checking for liveness make use of this construction to detect non-Zenoness [TYB05, Li09].

Zenoness problem

The Zenoness problem asks if there *exists a Zeno run* in a given timed automaton. Inspired by the strongly non-Zeno construction, a static check on the automaton has been proposed that ensures the absence of Zeno runs [GB07]. The method is efficient as it performs only static analysis: it looks at the automaton itself and not at its executions. But then, the method is not complete.

Contributions of the thesis

We focus on constructions for non-Zenoness and Zenoness in this part of the thesis. The contributions are listen in Table 1.2. We detail them below.

We observe that the apparently simple transformation to strongly non-Zeno automata [TYB05] can give a big overhead in the size of a simulation graph. We closely examine the transformation to strongly non-Zeno automata, and show that it can inflict a blowup of the simulation graph; and this blowup could even be exponential in the number of clocks, irrespective

of the abstraction used in the simulation graph. To substantiate, we exhibit an example of an automaton having a simulation graph of polynomial size, whose transformed version has a simulation graph of exponential size. Moreover, we see that this is true for all the abstractions used for the liveness problem.

We propose another solution to avoid this phenomenon. Instead of modifying the automaton, we modify the simulation graph. We show that this modification allows us to detect if a path in the simulation graph can be instantiated to a non-Zeno run. Moreover when we consider simulation graphs using some M-abstraction, the size of the modified graph is $|SG^M(\mathcal{A})| \cdot \mathcal{O}(|X|)$, where $|SG^M(\mathcal{A})|$ is the size of the simulation graph obtained from the M-abstraction under consideration and $|X|$ is the number of clocks. However we observe that the modified solution we propose still gives an exponential blowup for LU-abstractions. A further investigation reveals that this blowup is possibly unavoidable for LU-abstractions. We show that deciding non-Zenoness from simulation graphs using LU-abstractions is NP-complete. We propose a weakening of the LU-abstractions that avoids the exponential blowup required for the non-Zenoness check.

As a next contribution, we propose an on-the-fly algorithm for the Büchi non-emptiness problem. Our problem highly resembles the emptiness testing of finite automata with generalized Büchi conditions. Since the most efficient solutions for the latter problem are based on Tarjan's algorithm to detect strongly-connected-components (SCCs) [SE05, GS09], we take the same route here. We additionally observe that Büchi non-emptiness can sometimes be decided directly from the simulation graph. This permits to restrict the use of the modified simulation graph construction only to certain parts of the simulation graph. We also give additional optimizations that prove to be powerful in practice. We have implemented the algorithm in a prototype too. We report on experiments conducted on some standard examples in the literature.

Finally, we consider the Zenoness problem. As we have seen, the current solution gives a sufficient-only criterion for absence of Zeno runs. We propose a complete algorithm for the Zenoness problem by considering the simulation graph of the automaton. The solution constructs a graph that is twice as big as the simulation graph in the case of M-abstractions. However, it turns out that deciding Zenoness from simulation graphs using LU-abstractions is NP-complete. We observe the reason for this blowup and propose a weakening of the LU-abstractions that avoids this blowup.

Related work

The Büchi non-emptiness problem has been considered in the paper introducing timed automata [AD94]. The solution in this paper has been based on regions. This showed the decidability of the problem. The zone approach

to liveness was introduced by Tripakis *et al.* in [TYB05]. This paper is restricted to M-abstractions. In the same paper, they introduce the strongly non-Zenoness construction. A tool PROFOUNDER using M-abstractions and strongly non-Zenoness construction has been implemented [TYB05]. More recently, Li has shown that Büchi non-emptiness can be checked with every abstraction based on a time-abstract simulation [Li09]. This makes LU-abstractions correct for Büchi non-emptiness. For non-Zenoness, [Li09] still makes use of the strongly non-Zeno construction. This approach has been implemented in a tool CTAV [Li09].

Organization of the thesis

The thesis is divided into two parts: Part I on Reachability and Part II on Liveness. Each part contains a self sufficient Preliminaries chapter recalling the existing notions pertaining to the corresponding problem (Chapters 2 and 6). Chapters 3-5 talk about non-convex abstractions and on-the-fly computation of bounds for the reachability analysis. Chapters 7-9 focus on non-Zenoness and Zenoness problems. Experimental results are reported in Chapters 5 and 8. Each of these chapters ends with a section on concluding remarks, where a comprehensive summary of the chapter is provided with references to important definitions and results. The thesis ends with a Conclusion including perspectives for future work in Chapter 10.

Part I

Reachability

Chapter 2

Preliminaries

This chapter formally introduces timed automata and describes the standard approaches to solve the reachability problem. The main challenge in the analysis of timed automata is the uncountably infinite space of its configurations. One needs an effective way to handle this infinite space.

The first solution to the reachability problem partitions this infinite space of configurations into a finite number of *regions* [AD94], in such a way that configurations within a region are indistinguishable with respect to state reachability. Although this gives an effective procedure, the number of regions is too big to handle in practice.

A more efficient solution involves working with sets of clock valuations called *zones*. The automaton is explored for all its paths and at each step, all valuations reachable at the particular step are collected [DT98]. These collections can be represented by zones which in turn can be efficiently manipulated [Dil89, BY04]. The graph obtained thus is called the zone graph. This naive exploration may not terminate as the zone graph could potentially be infinite. One needs to consider the structure of the automaton and come up with abstractions of zones that are sound and complete for reachability.

Organization of the chapter

This chapter brings together the literature related to the reachability problem for timed automata. We begin with Section 2.1 in which we recall timed automata and define the reachability problem formally. We discuss the region approach in Section 2.2. Zones are handled in Section 2.3. The following Section 2.4 shows how one could abstract zones to get a finite graph capturing the behaviour of the potentially infinite zone graph. The same section also discusses what parameters are useful for defining the abstraction functions. We recall the standard abstractions existing in the literature in Section 2.5. In Section 2.6 we describe an optimization used to get better parameters for abstraction. Finally in Section 2.7 we present the state-

of-the-art algorithm for reachability testing. We end the chapter with an outline (Section 2.8) of our contributions to the reachability problem that are discussed in the following chapters.

2.1 Timed automata

Let $\mathbb{R}_{\geq 0}$ denote the set of non-negative reals. A clock is a variable that ranges over $\mathbb{R}_{\geq 0}$. Let X be a set of clocks. A *clock constraint* ϕ is a conjunction of comparisons of a clock with a constant, given by the following grammar:

$$\phi := x \sim c \mid \phi \wedge \phi$$

where $x \in X$, $\sim \in \{<, \leq, =, \geq, >\}$ and $c \in \mathbb{N}$. For example, $(x \leq 3 \wedge y > 0)$ is a clock constraint. Let $\Phi(X)$ denote the set of clock constraints over the set of clocks X.

A *clock valuation* over X is a function $v : X \to \mathbb{R}_{\geq 0}$. The set of all clock valuations is denoted by $\mathbb{R}_{\geq 0}^X$. We denote by $\mathbf{0}$ the valuation that associates 0 to every clock in X. A valuation v is said to satisfy a constraint ϕ, written as $v \vDash \phi$, when every constraint in ϕ holds after replacing every x by $v(x)$. For $\delta \in \mathbb{R}_{\geq 0}$, let $v + \delta$ be the valuation that associates $v(x) + \delta$ to every clock x. For $R \subseteq X$, let $[R]v$ be the valuation that sets x to 0 if $x \in R$, and that sets x to $v(x)$ otherwise. For a valuation v and a clock x, we denote the integral part of $v(x)$ by $\lfloor v(x) \rfloor$ and the fractional part of $v(x)$ by $\{v(x)\}$. We write $\lessdot$ to mean either $\leq$ or $<$, and $\gtrdot$ to mean either $\geq$ or $>$.

Definition 2.1.1 (Timed automaton [AD94]) A *Timed Automaton (TA)* is a tuple $\mathcal{A} = (Q, q_0, X, T, Acc)$ where

- Q is a finite set of states,
- $q_0 \in Q$ is the initial state,
- X is a finite set of clocks,
- $Acc \subseteq Q$ is a set of accepting states,
- $T \subseteq Q \times \Phi(X) \times 2^X \times Q$ is a finite set of transitions (q, g, R, q') where g is a clock constraint called the *guard*, and R is the set of clocks that are *reset* on the transition.

Figure 2.1 shows an example of a simple timed automaton. There are three states q_0, q_1 and q_2 and two clocks x, y. Clock x ensures that the automaton does not stay in state q_1 for more than 2 time units before coming back to q_0. Clock y ensures that the automaton can go to the final state q_2 only after 5 time units.

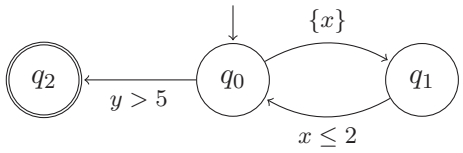


Figure 2.1: A timed automaton $\mathcal{A}_1$

Remark 2.1.2 The class of timed automata we consider is usually known as *diagonal-free* since clock comparisons like $x-y > 1$ are disallowed. A note about diagonal constraints is given in page 166 as part of the Conclusion.

Definition 2.1.3 (Semantics of a timed automaton) Let $\mathcal{A}$ be a timed automaton. The semantics of $\mathcal{A}$ is given by a transition system $\mathcal{S}_{\mathcal{A}}$ whose nodes are configurations (q, v) consisting of a state q of $\mathcal{A}$ and a valuation v giving the values of clocks. The initial configuration is given by $(q_0, \mathbf{0})$ with q_0 being the initial state of $\mathcal{A}$. The transition relation $\rightarrow$ is a union of two kinds of transitions:

delay $(q, v) \rightarrow^{\delta} (q, v + \delta)$ for some $\delta \in \mathbb{R}_{\geq 0}$;

action $(q, v) \rightarrow^{t} (q', v')$ for some transition $t = (q, g, R, q') \in T$ such that $v \vDash g$ and $v' = [R]v$.

A *run* of $\mathcal{A}$ is a finite sequence of transitions starting from the initial configuration $(q_0, \mathbf{0})$. Without loss of generality, we can assume that the first transition is a delay transition and that delay and action transitions alternate. We write $(q, v) \xrightarrow{\delta,t} (q', v')$ if there is a delay transition $(q, v) \rightarrow^{\delta} (q, v + \delta)$ followed by an action transition $(q, v + \delta) \rightarrow^{t} (q', v')$. So a run of $\mathcal{A}$ can be written as:

$$(q_0, v_0) \xrightarrow{\delta_0,t_0} (q_1, v_1) \xrightarrow{\delta_1,t_1} (q_2, v_2) \ldots (q_n, v_n)$$

where (q_0, v_0) represents the initial configuration $(q_0, \mathbf{0})$.

A run is *accepting* if it ends in a configuration (q_n, v_n) with $q_n \in Acc$.

Definition 2.1.4 (Reachability problem) The *reachability problem* for timed automata is to decide whether a given automaton has an accepting run. This problem is known to be PSPACE-complete [AD94, CY92].

Remark 2.1.5 In [HNSY94], a notion of timed automata with *state invariants* has been defined, in which every state is associated with a clock constraint of the form $x < c$ or $x \leq c$. This gives an upper bound on the amount of time that an automaton can spend at a state. Notice that if we are interested in state reachability, considering timed automata without

state invariants does not entail any loss of generality as the invariants can be added to the guards. For state reachability, we can also consider automata without transition labels.

Observe that the transition system $\mathcal{S}_{\mathcal{A}}$ has uncountably infinite configurations due to the delay transitions. Therefore, one cannot hope to work with $\mathcal{S}_{\mathcal{A}}$ in order to solve the reachability problem.

2.2 Regions

Since the transition system determined by an automaton is infinite, the standard solution is to find a finite approximation of this transition system by grouping valuations together. One such grouping is given by the *region abstraction* [AD94]. The space of valuations is partitioned into a finite number of *regions*. Two valuations within a region are indistinguishable with respect to reachability. Having formed these finite number of regions, a cross product with the states of the automaton is taken to give state-augmented regions. These state-augmented regions act as nodes of what is called the *region graph* of the automaton whose transitions are defined in a natural way, using the valuations present in the region. The analysis of the automaton is then performed using this finite region graph.

Let X be a finite set of clocks. Let $M : X \mapsto \mathbb{N} \cup \{-\infty\}$ be a *bound function* that associates a constant M_x to every clock x.

Definition 2.2.1 (Region equivalence [AD94]) Two valuations $v, v' \in \mathbb{R}^X_{\geq 0}$ are *region equivalent* w.r.t. M, denoted $v \sim_M v'$ iff for every $x, y \in X$:

1. $v(x) > M_x$ iff $v'(x) > M_x$;
2. if $v(x) \leq M_x$, then $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$;
3. if $v(x) \leq M_x$, then $\{v(x)\} = 0$ iff $\{v'(x)\} = 0$;
4. if $v(x) \leq M_x$ and $v(y) \leq M_y$ then $\{v(x)\} \leq \{v(y)\}$ iff $\{v'(x)\} \leq \{v'(y)\}$.

Given an automaton $\mathcal{A}$, a bound function is obtained by choosing for a clock x, the maximum constant appearing in a guard involving x. Then, the first three conditions in the above definition ensure that the two valuations satisfy the same guards. The last one enforces that for every $\delta \in \mathbb{R}_{\geq 0}$ there is $\delta' \in \mathbb{R}_{\geq 0}$, such that valuations $v + \delta$ and $v' + \delta'$ satisfy the same guards.

Definition 2.2.2 (Region [AD94]) Let $M : X \mapsto \mathbb{N} \cup \{-\infty\}$ be a bound function. A *region* is an equivalence class of $\sim_M$. We write $[v]^M$ for the region of v, and $\mathcal{R}_M$ for the set of all regions with respect to M.

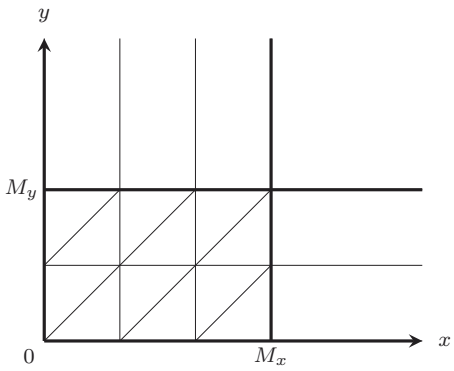


Figure 2.2: Division into regions with two clocks x and y [AD94].

Figure 2.2 shows the division into regions when there are two clocks x and y. We also give below a constructive definition of regions which would be useful in the later chapters.

Definition 2.2.3 (Region: constructive definition [AD94]) A region with respect to bound function M is a set of valuations specified as follows:

1. for each clock $x \in X$, one constraint from the set:
   $\{x = c \mid c = 0, \ldots, M_x\} \cup \{c-1 < x < c \mid c = 1, \ldots, M_x\} \cup \{x > M_x\}$
2. for each pair of clocks x, y having interval constraints: $c - 1 < x < c$ and $d - 1 < y < d$, it is specified if $\{x\}$ is less than, equal to or greater than $\{y\}$.

If r is a region then we will write $r \vDash g$ to mean that every valuation in r satisfies the guard g. It is straightforward to see that if a valuation $v \in r$ satisfies the guard g, then every valuation $v' \in r$ satisfies g. We will now show the other property with respect to time-elapse mentioned above.

Lemma 2.2.4 Let v, v' be valuations such that $v' \sim_M v$. Then, for all $\delta \in \mathbb{R}_{\geq 0}$, there exists a $\delta' \in \mathbb{R}_{\geq 0}$ such that $v' + \delta' \sim_M v + \delta$.

Proof
We know $v' \sim_M v$ and we are given δ. We need to choose δ'. Put $\lfloor \delta' \rfloor$ to be $\lfloor \delta \rfloor$. Clearly, we have $v' + \lfloor \delta' \rfloor \sim_M v + \lfloor \delta \rfloor$: that is, valuations $v' + \lfloor \delta' \rfloor$ and $v + \lfloor \delta \rfloor$ have the same integral parts and the same ordering of fractional parts (modulo M). Let $x_1 \preccurlyeq_1 x_2 \preccurlyeq_2 \ldots \preccurlyeq_{k-1} x_k$ be the ordering of fractional parts of clocks less than M in both the valuations. Here $\preccurlyeq$ denotes either $<$ or $=$.

From $v + \lfloor \delta \rfloor$, elapsing a fractional amount $\{\delta\}$ might move some of the clocks up to the next integer. Let $x_j, x_{j+1}, \ldots, x_k$ be the clocks that have their integral values increased from $v + \lfloor \delta \rfloor$ due to the fractional elapse $\{\delta\}$.

Thanks to the denseness of the real line, one can choose $\{\delta'\}$ between the fractional values of clocks x_{j-1} and x_j in $v' + \lfloor \delta' \rfloor$ so that $v' + \delta'$ has the same integers as $v + \delta$ and the same ordering of fractional parts (modulo M). □

Given a bound function M, the number of regions in $\mathcal{R}_M$ is finite. Once this finite partition of the valuations is obtained, we proceed to define a finite graph built from these regions, that captures the behaviour of the timed automaton.

For an automaton $\mathcal{A}$, to define its region graph, we consider a bound function $M_{\mathcal{A}}$ that is obtained from the automaton's definition.

Definition 2.2.5 (Maximal bounds) Given an automaton $\mathcal{A}$, the *maximal bounds function* $M_{\mathcal{A}} : X \mapsto \mathbb{N} \cup \{-\infty\}$ associates to each clock x the biggest constant appearing in a guard of the automaton that involves x. If there is no guard involving x, then $M_{\mathcal{A}}(x)$ is assigned $-\infty$.

We define the region graph of an automaton $\mathcal{A}$ using the $\sim_{M_{\mathcal{A}}}$ relation.

Definition 2.2.6 (Region graph [AD94]) Nodes of the *region graph* denoted by $RG(\mathcal{A})$ are of the form (q, r) for q a state of $\mathcal{A}$ and $r \in \mathcal{R}_{M_{\mathcal{A}}}$ a region. There is a transition $(q, r) \xrightarrow{t} (q', r')$ if there are $v \in r$, $\delta \in \mathbb{R}_{\geq 0}$ and $v' \in r'$ with $(q, v) \xrightarrow{\delta, t} (q', v')$. The initial node of the region graph is $(q_0, [\mathbf{0}]_{\sim_{M_{\mathcal{A}}}})$ where $[\mathbf{0}]_{\sim_{M_{\mathcal{A}}}}$ represents the region to which the initial valuation $\mathbf{0}$ belongs to. A node (q, r) is said to be an *accepting node* if $q \in Acc$.

Observe that a transition in the region graph is not decorated with a delay. Figure 2.3 shows a part of the region graph $RG(\mathcal{A}_1)$ of the automaton $\mathcal{A}_1$ shown in Figure 2.1.

It will be important to understand the property of pre-stability of regions [TYB05].

Lemma 2.2.7 (Pre-stability of regions) Let $\mathcal{A}$ be an automaton. Transitions in $RG(\mathcal{A})$ are pre-stable: in each transition $(q, r) \xrightarrow{t} (q', r')$, for every $v \in r$ there is a $\delta \in \mathbb{R}_{\geq 0}$ and a valuation $v' \in r'$ such that $(q, v) \xrightarrow{\delta, t} (q', v')$

Proof
By definition of the region graph, a transition $(q_1, r_1) \xrightarrow{t} (q_2, r_2)$ exists in $RG(\mathcal{A})$ if there are $v_1 \in r_1$, $\delta \in \mathbb{R}_{\geq 0}$ and $v_2 \in r_2$ with $(q_1, v_1) \xrightarrow{\delta, t} (q_2, v_2)$.

Let the transition t be (q_1, g, R, q_2). Pick a valuation $v_1' \in r_1$. By Lemma 2.2.4, there exists a δ' such that $v_1 + \delta$ and $v_1' + \delta'$ belong to the same region. We know that valuations within the same region satisfy the same guards. Therefore since $v_1 + \delta \vDash g$, we get that $v_1' + \delta' \vDash g$ too. From the

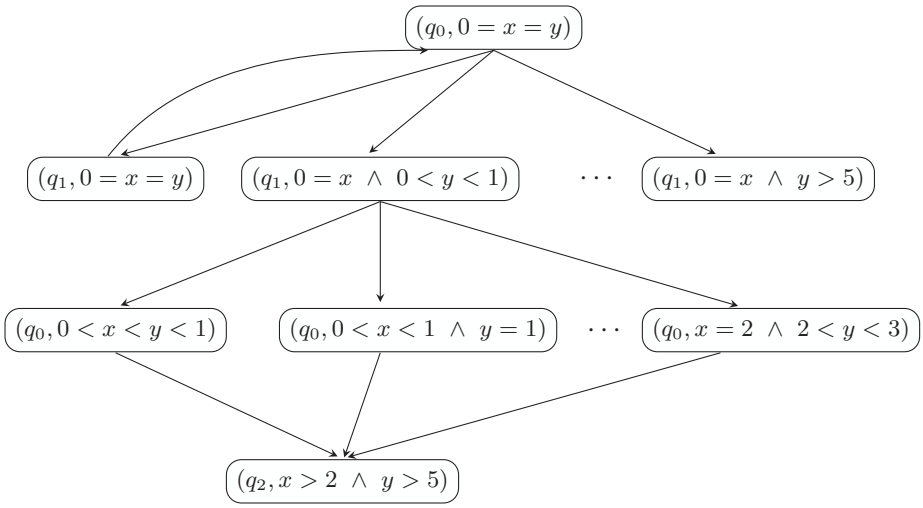


Figure 2.3: Part of region graph of the automaton $\mathcal{A}_1$ shown in Figure 2.1

definition of region equivalence, we get that regions are stable under projection to a subset of clocks and in particular, this entails that $[R](v_1' + \delta')$ belongs to the same region as $[R](v_1 + \delta)$. □

We will now establish the correspondence between paths of the region graph and runs of the automaton. Consider two sequences

$$(q_0, v_0) \xrightarrow{\delta_0, t_0} (q_1, v_1) \xrightarrow{\delta_1, t_1} \cdots (q_n, v_n) \tag{2.1}$$

$$(q_0, r_0) \xrightarrow{t_0} (q_1, r_1) \xrightarrow{t_1} \cdots (q_n, r_n) \tag{2.2}$$

where the first is a run in $\mathcal{A}$, and the second is a path in $RG(\mathcal{A})$. We say that the first is an *instantiation* of the second if $v_i \in r_i$ for all $i \in \{1, \ldots, n\}$. Equivalently, we say that the second is an *abstraction* of the first. The following lemma is a direct consequence of the pre-stability property.

Lemma 2.2.8 Every path in $RG(\mathcal{A})$ is an abstraction of a run of $\mathcal{A}$, and conversely, every run of $\mathcal{A}$ is an instantiation of a path in $RG(\mathcal{A})$.

The above lemma shows that the region graph is sound and complete for state reachability.

Theorem 2.2.9 ([AD94]) *Automaton $\mathcal{A}$ has an accepting run iff there is a path in the region graph $RG(\mathcal{A})$ starting from its initial node to an accepting node.*

While this theorem gives an algorithm for solving our problem, it turns out that this method is very impractical. The number of regions obtained using a bound function M is $\mathcal{O}\big(|X|! \cdot 2^{|X|} \cdot \prod_{x \in X}(2M_x + 2)\big)$ [AD94] and constructing all of them, or even searching through them on-the-fly, has proved to be very costly.

2.3 Zones

Although the region abstraction gives an effective procedure to solve the reachability problem, the size of the region graph obtained is too big to handle in practice. A more efficient solution is to group together all the valuations reaching a state of the automaton via a particular path. In consequence, we work with configurations consisting of a state and a set of valuations. We first define a transition relation $\Rightarrow$ over nodes of the form (q, W) where W is a set of valuations.

Definition 2.3.1 (Symbolic transition $\Rightarrow$) Let $\mathcal{A}$ be a timed automaton. For every transition t of $\mathcal{A}$ and every set of valuations W, we have a transition $\Rightarrow^t$ defined as follows:

$$(q, W) \Rightarrow^t (q, W') \text{ where } W' = \{v' \mid \exists v \in W,\ \exists \delta \in \mathbb{R}_{\geq 0}.\ (q, v) \rightarrow^t \rightarrow^\delta (q', v')\}$$

The transition relation $\Rightarrow$ is the union of all $\Rightarrow^t$.

The transition relation defined above considers each valuation $v \in W$ that can take the transition t, obtains the valuation after the transition and then collects the time-successors from this obtained valuation. Therefore the symbolic transition $\Rightarrow$ always yields sets closed under time-successors. The initial configuration of the automaton is $(q_0, \mathbf{0})$. Starting from the initial valuation $\mathbf{0}$ the set of valuations reachable by a time elapse at the initial state are given by $\{\mathbf{0} + \delta \mid \delta \in \mathbb{R}_{\geq 0}\}$. Call this W_0. From (q_0, W_0) as the initial node, computing the symbolic transition relation $\Rightarrow$ leads to different nodes (q, W) wherein the sets W are closed under time-successors.

Remark 2.3.2 Our choice of semantics in the symbolic transition $\Rightarrow$ of Definition 2.3.1 differs from some literature [Bou09, TYB05] in the order of time and action transitions. We explain our choice of semantics in Appendix A.

It has additionally been noticed that the sets W obtained in the nodes (q, W) can be described by some simple constraints involving only the difference between clocks [BY04]. This has motivated the definition of *zones*, which are sets of valuations defined by difference constraints.

Definition 2.3.3 (Zones [BY04]) A zone is a set of valuations defined by a conjunction of two kinds of clock constraints: for $x, y \in X$

$$x \sim c$$
$$x - y \sim c$$

where, $\sim \in \{\leq, <, =, >, \geq\}$ and $c \in \mathbb{Z}$. For example, $(x > 4 \ \wedge \ y - x \leq 1)$ is a zone.

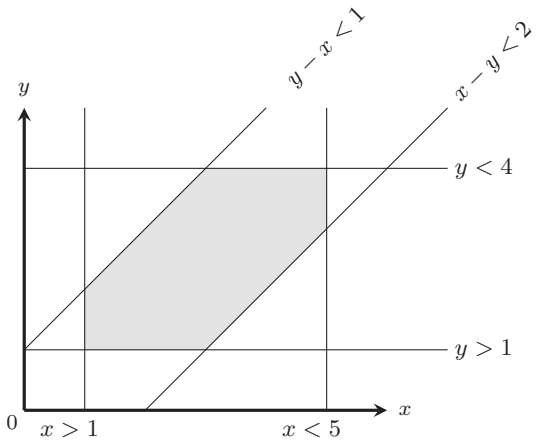


Figure 2.4: An example of a zone

Another example of a zone is illustrated in Figure 2.4. It can be shown that starting from a node (q, W) with W being a zone, the transition $(q, W) \Rightarrow (q', W')$ leads to a node in which W' is again a zone [BY04]. Observe that the initial set of valuations $W_0 = \{\mathbf{0} + \delta \mid \delta \in \mathbb{R}_{\geq 0}\}$ is indeed a zone: it is given by the constraints

$$\bigwedge_{x,y \in X} (x \geq 0 \ \wedge \ x - y = 0)$$

We will now define a symbolic semantics of timed automata which is a transition system with nodes consisting of zones. This is called the *zone graph* of the automaton. In the sequel, zones are denoted by Z, Z', etc.

Definition 2.3.4 (Zone graph) Given a timed automaton $\mathcal{A} = (Q, q_0, X, T, Acc)$, the *zone graph* $ZG(\mathcal{A})$ of $\mathcal{A}$ is a transition system whose nodes are of the form (q, Z) with $q \in Q$ and Z a zone. The initial node is (q_0, Z_0) where $Z_0 = \{\mathbf{0} + \delta \mid \delta \in \mathbb{R}_{\geq 0}\}$ is the set of valuations obtained by elapsing time from $\mathbf{0}$. The transitions are given by the relation $\Rightarrow$ of Definition 2.3.1.

A part of the zone graph for the automaton of Figure 2.1 is shown in Figure 2.5. Compare the zone graph to the region graph shown in Figure 2.3. There is a marked decrease in the size of the graph.

Representing zones

We will now see how to represent zones and compute the successors in the zone graph efficiently. We start by describing how to represent zones efficiently. The example of a zone over two clocks x, y in Figure 2.4 gets defined by six constraints. Now let us look at a more abstract notation. A zone over two clocks can be defined by six constraints as shown in Figure 2.6. Due to

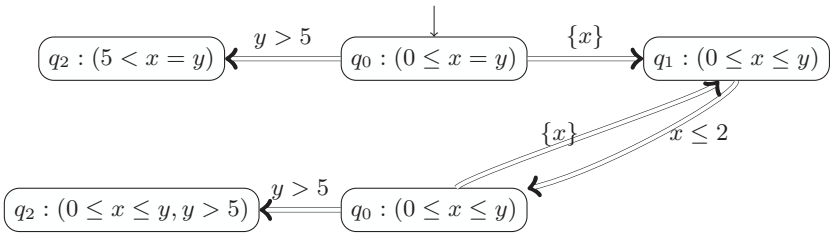


Figure 2.5: $ZG(\mathcal{A}_1)$ for the automaton in Figure 2.1

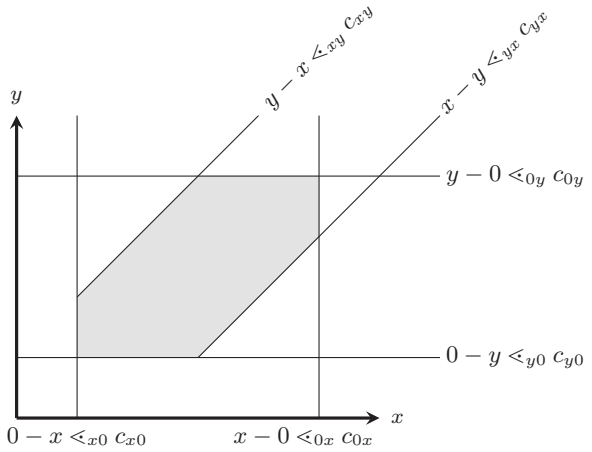


Figure 2.6: An arbitrary zone over two clocks x, y

this specific structure, an arbitrary zone can be given by a Difference-Bound Matrix (DBM) [Dil89].

Definition 2.3.5 (Matrix representation) Let X be a set of clocks. A special variable x_0 representing the constant 0 is added to the set X. A zone is represented by a matrix $(Z_{xy})_{x,y \in X}$ in which each entry $Z_{xy} = (\prec_{xy}, c_{xy})$ represents the constraint $y - x \prec c_{xy}$ where $c_{xy} \in \mathbb{Z}$ and $\prec \in \{\leq, <\}$ or $(\prec_{xy}, c_{xy}) = (<, \infty)$. For convenience we sometimes write 0 to represent the variable x_0.

The above definition with a slight variation is the DBM representation of a zone. In the standard DBM notation, the entry $Z_{xy} = (\prec_{xy}, c_{xy})$ stands for the constraint $x - y \prec_{xy} c_{xy}$. We however choose to do it the other way: $y - x \prec_{xy} c_{xy}$. This is because later on in Chapter 4, we consider a *graph representation* of a zone wherein the edge from x to y with weight $(\prec_{xy}, c_{xy})$ represents the constraint $y - x \prec_{xy} c_{xy}$. The bottom-line for readers familiar with the standard DBM notation:

$$Z_{xy} = (\prec_{xy}, c_{xy}) \text{ denotes } y - x \prec_{xy} c_{xy}$$

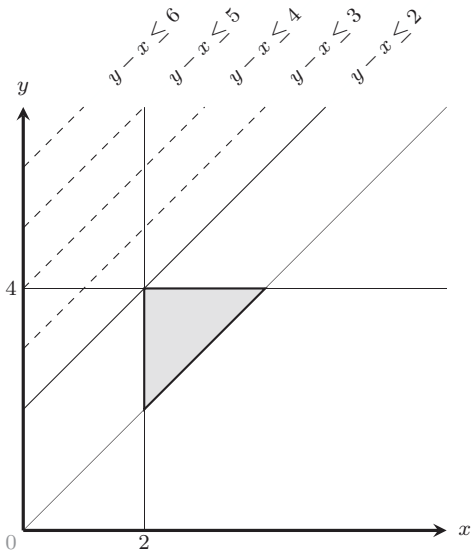


Figure 2.7: The $y - x \lessdot_{xy} c_{xy}$ constraint for the zone gets defined by the constraints $x \geq 2$ and $y \leq 4$. Adding the constraint $y - x \leq c$ for any $c \geq 2$ still gives the same zone

in this thesis.

Consider the zone showed in Figure 2.7. It is defined by the constraints:

$$(\ x \geq 2 \ \wedge \ y \leq 4 \ \wedge \ x - y \leq 0 \ )$$

If one adds $y - x \leq 6$ to the above system of constraints, the solution set does not change. In fact, one could add any constraint $y - x \leq c$ for $c \geq 2$ and the solution set remains the same. This is because the diagonal constraint gets defined by $x \geq 2$ and $y \leq 4$. This is depicted in Figure 2.7. Adding the constraint given by the dashed diagonals to the set of constraints given above does not change the solution set. However, one cannot go below the diagonal $y - x \leq 2$, as doing so would alter the solution set. It is a *tight* constraint.

Every zone can be represented by a *canonical* DBM. A canonical DBM is one in which reducing the weight $(\lessdot_{xy}, c_{xy})$ would change the solution set. In other words, each constraint is tight in the canonical representation. Given a DBM with n rows and columns, the canonical DBM representing the same solution set can be calculated in time $\mathcal{O}(n^3)$ using Floyd-Warshall's algorithm for shortest paths [BY04].

Successor computation in the zone graph

The successor computation $(q, Z) \Rightarrow^t (q', Z')$ for a transition $t = (q, g, R, q')$ proceeds in the following steps.

Operation	Complexity	
$Z \wedge g$	$\mathcal{O}(\|X\|^2)$	$Z \wedge g = \{v \mid v \in Z \text{ and } v \vDash g\}$
$[R](Z)$	$\mathcal{O}(\|X\|.\|R\|)$	$[R](Z) = \{[R]v \mid v \in Z\}$
elapse(Z)	$\mathcal{O}(\|X\|)$	elapse$(Z) = \{v + \delta \mid v \in Z \text{ and } \delta \in \mathbb{R}_{\geq 0}\}$
$Z \subseteq Z'$	$\mathcal{O}(\|X\|^2)$	

Table 2.1: Operations on zones (**note that g is diagonal free)**

$$(q, Z) \xrightarrow{guard} (q, Z \wedge g) \xrightarrow{reset} (q, [R](Z \wedge g)) \xrightarrow{elapse} (q, Z')$$

In the above, $Z \wedge g$ represents the set of valuations that satisfy the constraints of both Z and g; the set $[R](Z \wedge g)$ represents the set of valuations obtained by resetting clocks in R from every valuation in $Z \wedge g$ and finally Z' is the set of valuations obtained by elapsing an arbitrary amount of time from $[R](Z \wedge g)$. All these operations can be computed efficiently using DBMs. At each step, the canonical DBM representing the zone obtained is computed.

The costliest operation is the computation of the intersection of a zone with a guard. In the general case when the guards are diagonals like $x - y \leq 5$, the intersection takes $\mathcal{O}(|X|^3)$ time due to the canonicalization procedure. However when the guards are diagonal free, it has been shown in [ZLZ05] that the canonicalization is easier and hence the intersection operation can be done in $\mathcal{O}(|X|^2)$ time. Another crucial operation required in algorithms using zones is to know when a zone Z is included in another zone Z'. We list the common operations on zones and the complexity required to perform these operations in Table 2.1. We direct the reader to [BY04] for more details on implementing these operations using DBMs. From the table it can be inferred that computing the successor in the zone graph has a complexity quadratic in the number of clocks.

For an automaton, it appears that its zone graph is a more succinct representation of its behaviour as compared to its region graph. Therefore, it might be desirable to work with the zone graph instead of the region graph. But, it is not a pretty picture as yet.

The zone graph could be infinite

Consider the automaton $\mathcal{A}_{inf}$ shown in Figure 2.8, with two clocks $\{x, y\}$ and no accepting state. The initial node is given by $(q_0, x = y \ \wedge \ x \geq 0)$. The transition to q_1 gives the node $(q_1, x = y \ \wedge \ x \geq 0)$. The only transition from q_1 taken from this node gives the node $(q_1, x - y = 1 \ \wedge \ x \geq 0)$, which is a new node. This node has its own successors and the process continues.

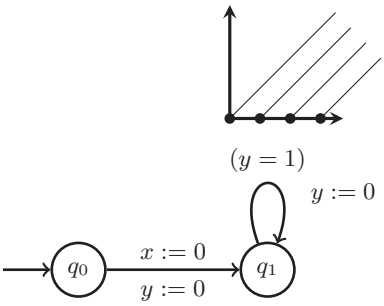


Figure 2.8: Automaton $\mathcal{A}_{inf}$ and the graph of zones obtained at q_1

Finally at q_1 we have the following zones in the zone graph $ZG(\mathcal{A}_{inf})$:

$$(\ x - y = k \ \wedge \ x \geq 0 \ ) \ \text{ for all } k \in \mathbb{N}$$

This is pictorially shown in Figure 2.8.

A standard solution to this problem of non-termination is to use *abstractions*.

2.4 Bounds and abstractions

In the previous section, we defined a symbolic semantics for timed automata called the zone graph. Although the zone graph could be much smaller than the region graph in a number of cases, it could potentially be infinite. Therefore, we usually try to find a finite approximation of it by further grouping valuations together. An *abstraction operator* is a convenient way to express a grouping of valuations.

Definition 2.4.1 (Abstraction operator [BBFL03]) Let W be a set of valuations. An *abstraction operator* is a function $\mathfrak{a} : \mathcal{P}(\mathbb{R}^{|X|}_{\geq 0}) \to \mathcal{P}(\mathbb{R}^{|X|}_{\geq 0})$ such that $W \subseteq \mathfrak{a}(W)$ and $\mathfrak{a}(\mathfrak{a}(W)) = \mathfrak{a}(W)$.

If $\mathfrak{a}$ has a finite range then this abstraction is finite. Note that the abstraction $\mathfrak{a}(Z)$ of a zone Z is in general an arbitrary set, and not necessarily a zone. An abstraction operator $\mathfrak{a}$ defines an abstract semantics. We call this abstract semantics the *simulation graph* of the automaton.

Definition 2.4.2 (Simulation graph)[1] Let $\mathcal{A}$ be an automaton. Given an abstraction operator $\mathfrak{a}$, the *simulation graph* $SG^{\mathfrak{a}}(\mathcal{A})$ is the transition system with nodes of the form (q, W) where q is a state of the automaton and W is a set of valuations. The initial node of $SG^{\mathfrak{a}}(\mathcal{A})$ is $(q_0, \mathfrak{a}(W_0))$

[1] In the terminology of [TYB05], a simulation graph is more generic and includes any symbolic semantics using sets of valuations. The zone graph that we define is called the exact simulation graph in [TYB05].

where $W_0 = \{\mathbf{0} + \delta \mid \delta \in \mathbb{R}_{\geq 0}\}$. There is a transition $(q, W) \Rightarrow_{\mathfrak{a}} (q', \mathfrak{a}(W'))$ when $W = \mathfrak{a}(W)$ and $(q, W) \Rightarrow (q', W')$ is a symbolic transition (c.f. Definition 2.3.1).

We want a simulation graph to reflect some properties of the original system. Let $\rightarrow^*$ denote the reflexive and transitive closure of the transition relation $\rightarrow$ on the concrete semantics $\mathcal{S}_{\mathcal{A}}$ defined in Definition 2.1.3. Similarly, let $\Rightarrow^*_{\mathfrak{a}}$ denote the reflexive and transitive closure of $\Rightarrow_{\mathfrak{a}}$ defined on the abstract semantics $SG^{\mathfrak{a}}(\mathcal{A})$. In order to preserve reachability properties we can require the following two properties: (W_0 denotes $\{\mathbf{0} + \delta \mid \delta \in \mathbb{R}_{\geq 0}\}$)

Soundness: if $(q_0, W_0) \Rightarrow^*_{\mathfrak{a}} (q, W)$ then there is $v \in W$ such that $(q_0, \mathbf{0}) \rightarrow^* (q, v)$.

Completeness: if $(q_0, \mathbf{0}) \rightarrow^* (q, v)$ then there is W such that $v \in W$ and $(q_0, W_0) \Rightarrow^*_{\mathfrak{a}} (q, W)$.

It can be easily verified that if an abstraction satisfies $W \subseteq \mathfrak{a}(W)$ then the abstracted system is complete. However soundness is more delicate to obtain. Before proceeding to look for sound abstractions, we highlight an important remark about the size of the simulation graph with respect to different abstractions. The coarser the abstraction, the smaller is the simulation graph.

Remark 2.4.3 If $\mathfrak{a}$ and $\mathfrak{b}$ are two abstractions such that for every set of valuations W, we have $\mathfrak{a}(W) \subseteq \mathfrak{b}(W)$, we prefer to use $\mathfrak{b}$ since the graph induced by it is *a priori* smaller than the one induced by $\mathfrak{a}$ (sic. [BBLP06]).

Therefore, the aim is to come up with a finite abstraction as coarse as possible, that still maintains the soundness property.

Abstractions from simulations

One way to obtain abstractions is to group together valuations that are not distinguishable by an automaton, i.e. consider a bisimulation relation. If we are after reachability properties it has been noted in [BBLP06] that one can even consider (time abstract) simulation relation [TAKB96]. For the rest of the section, assume a given automaton $\mathcal{A}$.

Definition 2.4.4 (Time-abstract simulation) A *(state based) time-abstract simulation* between two states of transition system $\mathcal{S}_{\mathcal{A}}$ is a relation $(q, v) \preceq_{t.a.} (q', v')$ such that:

- $q = q'$,
- if $(q, v) \rightarrow^{\delta} (q, v + \delta) \rightarrow^{t} (q_1, v_1)$, then there exists a $\delta' \in \mathbb{R}_{\geq 0}$ such that $(q, v') \rightarrow^{\delta'} (q, v' + \delta') \rightarrow^{t} (q_1, v'_1)$ satisfying $(q_1, v_1) \preceq_{t.a.} (q_1, v'_1)$ for the same transition t.

For two valuations v, v', we say that $v \preceq_{t.a.} v'$ if for every state q of the automaton, we have $(q, v) \preceq_{t.a.} (q', v')$. An abstraction $\mathfrak{a}_{\preceq_{t.a.}}$ based on a simulation $\preceq_{t.a.}$ can be defined as follows:

Definition 2.4.5 (Abstraction based on simulation) Given a set W, we define $\mathfrak{a}_{\preceq_{t.a.}}(W) = \{v \mid \exists v' \in W.\ v \preceq_{t.a.} v'\}$.

By definition of the simulation relation, an abstraction based on simulation is sound. For a given automaton it can be computed if two configurations are in a simulation relation. It should be noted though that computing the coarsest simulation relation is EXPTIME-hard [LS00]. Since the reachability problem can be solved in PSPACE, this suggests that it may not be reasonable to try to solve it using the abstraction based on the coarsest simulation. We can get simulation relations that are computationally easier if we consider only a part of the structure of the automaton. The common way is to look at constants appearing in the guards of the automaton and consider them as parameters for abstraction.

Parameters for abstraction

There are two kinds of parameters used for defining abstractions: the first is a bound function M that associates to every clock x the maximum constant that appears in a guard involving x as in Definition 2.2.5; more refined is to take the maximum separately over constants from lower bound constraints, that is in guards of the form $x > c$ or $x \geq c$, and those from upper bound constraints, that is in guards $x < c$ or $x \leq c$, giving two bound functions L and U respectively. If one moreover does this for every clock x separately, one gets for each clock two integers L_x and U_x.

Definition 2.4.6 (LU-bounds) The L bound for an automaton $\mathcal{A}$ is the function assigning to every clock x a maximal constant that appears in a lower bound guard for x in $\mathcal{A}$, that is, maximum over guards of the form $x > c$ or $x \geq c$. Similarly U is the function assigning to every clock x a maximal constant appearing in an upper bound guard for x in $\mathcal{A}$, that is, maximum over guards of the form $x < c$ or $x \leq c$.

The maximal bounds (M-bounds) of Definition 2.2.5 is then obtained by assigning to every clock x the maximum over the constants assigned by L and U, that is to say, for every clock x, we have $M_x = \max(L_x, U_x)$.

Convexity

We will now point out an important additional criterion expected from an abstraction operator. The whole purpose of finding a suitable abstraction $\mathfrak{a}$ is to make use of it to build the simulation graph $SG^{\mathfrak{a}}(\mathcal{A})$ for an automaton.

Naturally, it is important to be able to efficiently compute the simulation graph. We have seen that zones can be represented and manipulated efficiently. So in addition to the requirements of soundness and completeness, an abstraction is required to be *effective*, that is the abstraction of a zone is always a zone [Bou04].

Definition 2.4.7 (Convex abstractions) An abstraction operator $\mathfrak{a}$ is said to be *convex* if for all zones Z, the set $\mathfrak{a}(Z)$ is also a zone. Since a zone is a convex set, the abstraction ranges over convex sets and hence the name.

In the next section, we will see some specific abstraction operators existing in the literature.

2.5 Abstractions in literature

The solution to the reachability problem proceeds by computing the abstract semantics, also known as the simulation graph, using a suitable finite abstraction operator. As discussed in the previous section, an abstraction operator is required to be sound, complete and additionally efficiently representable. We have seen that abstractions based on simulation are sound by definition. To find a suitable simulation, one could use the LU-bounds or M-bounds. We have also noted that completeness of an abstraction is immediate as by definition of an abstraction operator $\mathfrak{a}$, we have $W \subseteq \mathfrak{a}(W)$ for all sets W. For efficiency, the abstraction is required to be convex.

With all these criteria in mind, there have been different abstraction operators defined in the literature. One can broadly divide these operators into two kinds, LU-abstractions which use LU-bounds as parameters and M-abstractions which use M-bounds as parameters.

LU-abstractions

The paper introducing LU-bounds [BBLP06] also introduced three abstraction operators: Extra_{LU}, Extra^{+}_{LU} and $\mathfrak{a}_{\preccurlyeq LU}$. Abstraction $\mathfrak{a}_{\preccurlyeq LU}$ is the coarsest among the three, but it could lead to non-convex sets. To circumvent this problem, abstractions Extra_{LU} and Extra^{+}_{LU}, the finer and convex versions of $\mathfrak{a}_{\preccurlyeq LU}$, were introduced.

We begin by recalling the definition of an LU-preorder defined in [BBLP06]. We use a different but equivalent formulation.

Definition 2.5.1 (LU-preorder [BBLP06]) Let $L, U : X \to \mathbb{N} \cup \{-\infty\}$ be two bound functions. For a pair of valuations we set $v \preccurlyeq_{LU} v'$ if for every clock x:

- if $v'(x) < v(x)$ then $v'(x) > L_x$, and

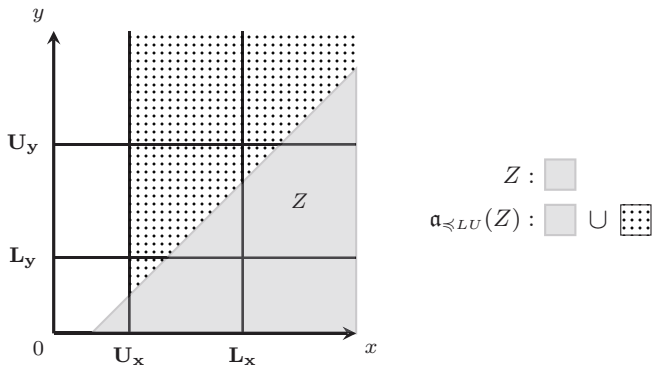


Figure 2.9: Zone Z is given by the grey area. Abstraction $\mathfrak{a}_{\preccurlyeq_{LU}}(Z)$ is given by the grey area along with the dotted area

- if $v'(x) > v(x)$ then $v(x) > U_x$.

It has been shown in [BBLP06] that $\preccurlyeq_{LU}$ is a simulation relation. The $\mathfrak{a}_{\preccurlyeq_{LU}}$ abstraction is based on this LU-preorder $\preccurlyeq_{LU}$.

Definition 2.5.2 ($\mathfrak{a}_{\preccurlyeq_{LU}}$-abstraction [BBLP06]) Given L and U bound functions, for a set of valuations W we define:

$$\mathfrak{a}_{\preccurlyeq_{LU}}(W) = \{v \mid \exists v' \in W.\ v \preccurlyeq_{LU} v'\}.$$

Figure 2.9 gives an example of a zone Z and its abstraction $\mathfrak{a}_{\preccurlyeq_{LU}}(Z)$. It can be seen that $\mathfrak{a}_{\preccurlyeq_{LU}}(Z)$ is not a convex set. We will now define its convex counterparts $Extra_{LU}$ and $Extra^{+}_{LU}$, also known as LU-extrapolations. For every zone Z, we will have $Extra_{LU}(Z) \subseteq Extra^{+}_{LU}(Z) \subseteq \mathfrak{a}_{\preccurlyeq_{LU}}(Z)$. As $\mathfrak{a}_{\preccurlyeq_{LU}}(Z)$ is sound, the LU-extrapolations are sound too.

We give a brief intuition to the definition of the LU-extrapolations. We know that a zone can be given by a set of constraints (c.f. Definition 2.3.5). To define an abstraction that gives back a zone, one could say that the abstraction operator takes each constraint $y-x \prec c$ of the zone and transforms it to some $y-x \prec' c'$. This would yield a zone again. The LU-extrapolations are defined keeping this in mind. However, it is required that the LU-extrapolation of a zone Z is always contained in $\mathfrak{a}_{\preccurlyeq_{LU}}(Z)$. Let us now see how it is achieved. Consider a constraint $y - x \leq c$ defining a zone Z. Therefore, there is a valuation $v' \in Z$ that satisfies $v'(y) - v'(x) = c$.

Suppose $c > L_y$. As $v'(y) = v'(x) + c$ and $v'(x) \geq 0$ by definition, this gives $v'(y) > L_y$. From the definition of the LU-preorder Definition 2.5.1, all valuations v such that $v(y) > v'(y)$ and $v(x) = v'(x)$ will satisfy $v \preccurlyeq_{LU} v'$. Hence one can safely add these valuations v to an abstraction of Z. After adding all these valuations, the obtained set no longer has an upper bound on $y-x$. One could therefore let $Extra_{LU}$ to change $y-x \leq c$ to $y-x < \infty$.

We stay within the limits of $\mathfrak{a}_{\preccurlyeq LU}$ and additionally maintain convexity. This is depicted in Figure 2.10. Here $Extra_{LU}$ works the same way as $\mathfrak{a}_{\preccurlyeq LU}$.

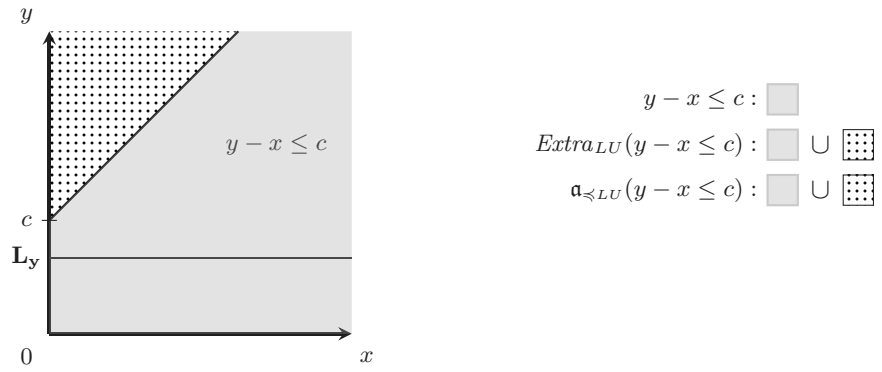


Figure 2.10: Zone has $y-x \leq c$ with $c > L_x$, then $Extra_{LU}$ changes it to $y-x < -\infty$

Now, consider the situation when $-c > U_x$. Figure 2.11 depicts the set of valuations $y - x \leq c$ and the $\mathfrak{a}_{\preccurlyeq LU}$ abstraction applied on it. The grey portion shows the valuations satisfying $y - x \leq c$ and the dotted portion along with the cross lines shows the valuations that are added by $\mathfrak{a}_{\preccurlyeq LU}$. Note that we cannot choose one value c' such that $y - x \leq c'$ adds exactly the valuations added by $\mathfrak{a}_{\preccurlyeq LU}$. So the $Extra_{LU}$ operator chooses to add only a subset of it by moving the constraint up to $y - x < -U_x$. This is shown by the patch drawn with cross lines. Here we see that $Extra_{LU}$ does not include all valuations of $\mathfrak{a}_{\preccurlyeq LU}$, but the transformation maintains convexity.

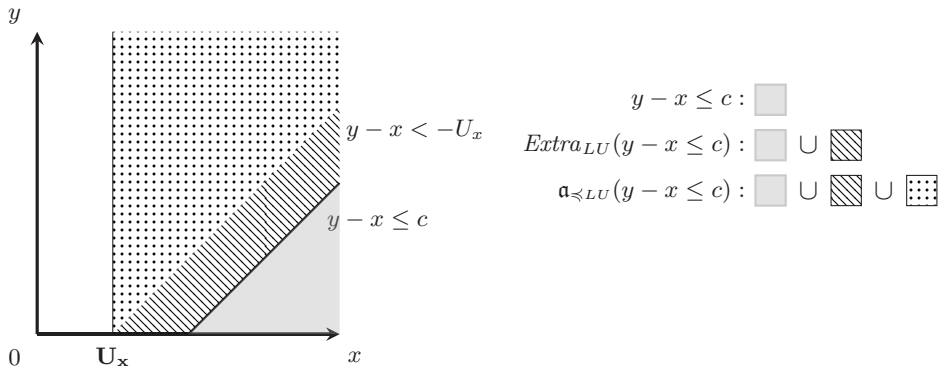


Figure 2.11: Zone has $y - x \leq c$ with $-c > U_x$, then $Extra_{LU}$ changes it to $y - x < -U_x$

Observe that both the definitions are a modification of a constraint $y-x$ to some other value. This way, the two cases discussed above maintain convexity: given a zone, $Extra_{LU}$ gives a zone again. $Extra^{+}_{LU}$ is an optimization of $Extra_{LU}$ which notes that if $y > L_y$ for all valuations in Z, then irrespective of the value of c, the constraint $y - x < c$ can be removed. Similarly

when $x > U_x$ for all valuations in Z, then $y - x < c$ can be changed to $y - x < U_x$.

We will now formally define $Extra_{LU}$ and $Extra^+_{LU}$ for a zone Z.

Definition 2.5.3 (LU-extrapolations) For a zone Z given by $(Z_{xy})_{x,y\in X}$ with $Z_{xy} = (\preccurlyeq, c_{xy})$, let Z^{LU} denote $Extra_{LU}(Z)$ and let Z^{LU+} denote $Extra^+_{LU}(Z)$. Then:

$$Z^{LU}_{xy} = \begin{cases} (\infty, <) & \text{if } c_{xy} > L(y) \\ (-U(x), <) & \text{if } -c_{xy} > U(x) \\ Z_{xy} & \text{otherwise} \end{cases}$$

$$Z^{LU+}_{xy} = \begin{cases} (\infty, <) & \text{if } c_{xy} > L(y) \\ (\infty, <) & \text{if } -c_{y0} > L(y) \\ (\infty, <) & \text{if } -c_{x0} > U(x), i \neq 0 \\ (-U(x), <) & \text{if } -c_{x0} > U(x), i = 0 \\ Z_{xy} & \text{otherwise} \end{cases}$$

where $L(x_0) = U(x_0) = 0$ for the special clock x_0.

M-abstractions

There are three M-abstractions defined: $Closure_M$, $Extra_M$ and $Extra^+_M$. The abstractions $Extra_M$ and $Extra^+_M$ are obtained respectively from $Extra_{LU}$ and $Extra^+_{LU}$ by setting both L and U to be the maximum bounds M. The $Closure_M$ abstraction of a set W is defined to be the union of regions w.r.t $\sim_M$ intersecting W. However, we will later see in Section 3.4 of Chapter 3 that $Closure_M$ can be obtained from $\mathfrak{a}_{\preccurlyeq LU}$ by substituting $L = U = M$.

Definition 2.5.4 (M-extrapolations) For a zone Z given by $(Z_{xy})_{x,y\in X}$ with $Z_{xy} = (\preccurlyeq, c_{xy})$, let Z^M denote the zone $Extra_M(Z)$ and Z^{M+} denote $Extra^+_M(Z)$. Then:

$$Z^{M}_{xy} = \begin{cases} (\infty, <) & \text{if } c_{xy} > M(y) \\ (-M(x), <) & \text{if } -c_{xy} > M(x) \\ Z_{xy} & \text{otherwise} \end{cases}$$

$$Z^{M+}_{xy} = \begin{cases} (\infty, <) & \text{if } c_{xy} > M(y) \\ (\infty, <) & \text{if } -c_{y0} > M(y) \\ (\infty, <) & \text{if } -c_{x0} > M(x), i \neq 0 \\ (-M(x), <) & \text{if } -c_{x0} > M(x), i = 0 \\ Z_{xy} & \text{otherwise} \end{cases}$$

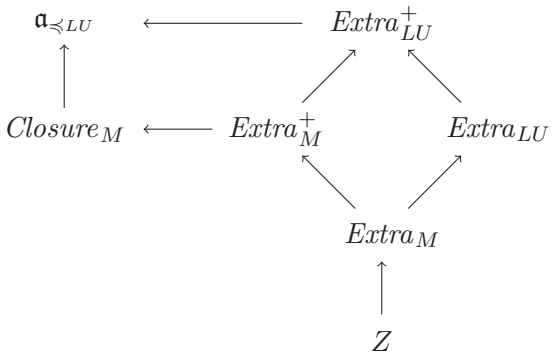


Figure 2.12: Hierarchy of abstractions in literature [BBLP06]. The ones starting with *Extra* are convex. $Extra^+_{LU}$ is currently used in implementations.

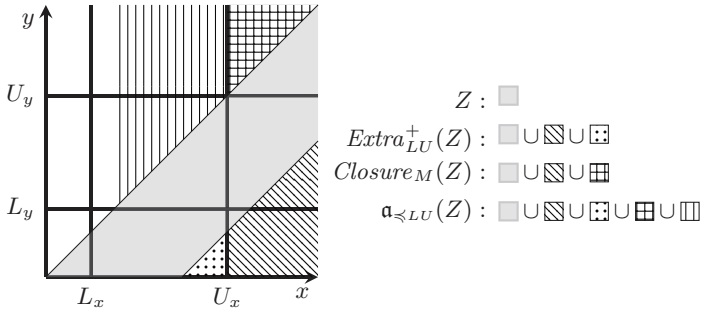


Figure 2.13: Comparing abstractions defined in Figure 2.12

where $M(x_0) = 0$ for the special clock x_0.

Definition 2.5.5 (Closure abstraction) For a zone Z, the abstraction $Closure_M(Z)$ is the union of all regions intersecting Z:

$$Closure_M(Z) = \bigcup \{ [v]^M \mid v \in Z \}$$

Figure 2.12 depicts the hierarchy of abstraction operators with respect to inclusion. The diamond on the right hand side shows the four extrapolations. As $Extra^+_{LU}$ is the coarsest among convex abstractions, it is currently used in state-of-the-art implementations like UPPAAL [BBLP06].

A comparison of the different abstractions is shown in Figure 2.13. We depict only $Extra^+_{LU}$, $Closure_M$ and $\mathfrak{a}_{\preccurlyeq LU}$ for clarity. Recall that bound function M is the maximum over L and U. The figure illustrates that $Extra^+_{LU}$ and $Closure_M$ are incomparable. Also note that $\mathfrak{a}_{\preccurlyeq LU}$ subsumes the other two.

2.6 Optimizing bounds by static analysis

In the previous section, we have seen that abstractions using LU-bounds are coarser and are used in implementations. Recall Remark 2.4.3 which states the importance of having coarser abstractions. Although $Extra^+_{LU}$ is the coarsest abstraction used in implementations, one could get better by providing better LU-bounds to $Extra^+_{LU}$. The smaller the bounds, the bigger is the zone returned by $Extra^+_{LU}$. A naive way to choose L bounds is to take for each clock the maximum constant appearing in a guard anywhere in the automaton that lower-bounds the clock. Similarly we choose U from among the upper bounded guards that occur anywhere in the automaton. In this section, we will see an improved approach to choose bounds.

Consider the automaton $\mathcal{A}_{stat}$ shown in Figure 2.14. The naive approach to choosing bounds will give $L_x = U_x = 10^6$, $L_y = U_y = 10^6$.

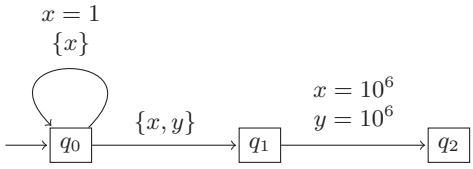


Figure 2.14: Timed automaton $\mathcal{A}_{stat}$.

As there are no accepting states, the reachability algorithm, even the one using the $Extra^+_{LU}$ abstraction needs to explore around 10^6 nodes. Thanks to the loop around q_0, one gets zones given by $y - x = k$ for $k \in \{1, \ldots, 10^6\}$. Clearly, no two of these zones are included with respect to $Extra^+_{LU}$ due to the large constants and the reachability algorithm needs to explore all of them.

Consider two configurations of $\mathcal{A}_{stat}$, given by $s_1 := (q_0, x = 10^7, y = 5)$ and $s_2 := (q_0, x = 10, y = 5)$. These two configurations are distinguished since the values of x are on different sides of the constant 10^6. However, note that the constant 10^6 appears only in the transition $q_1 \to q_2$. The only path to q_1 is the edge $q_0 \to q_1$ in which both the clocks are reset. So both the configurations on taking the edge reach the configuration $(q_1, x = 0, y = 0)$. There is no necessity to distinguish between s_1 and s_2. This has motivated the need for a better approach to assign bound functions.

It has been observed in [BBFL03] that instead of considering global bound functions LU for all states in an automaton, one can use different bound functions for each state of the automaton. We write LU to denote the two bound functions L and U. For every state q of an automaton, there are bound functions $LU(q)$ and in the simulation graph using $Extra^+_{LU}$, we have nodes of the form $(q, Extra^+_{LU(q)}(Z))$

At each state q, the bound function $L(q)$ is calculated as follows. We consider paths of the automaton starting from q. If there is a path from

q that contains a guard $x \geqslant c$ and the clock x is not reset in any edge till this guard, then the constant c is relevant. If we denote by $L_x(q)$ the constant assigned by $L(q)$ to x, then $L_x(q)$ is the maximum over such relevant constants.

$$q \to \underbrace{q_1 \to \ldots \to q_n}_{\text{no resets of } x} \xrightarrow{x \geqslant c} q'$$

The constant $U_x(q)$ is assigned similarly, but now looking at constraints of the form $x \leqslant c$.

Formally, for every state q and every clock x, there are constants $L_x(q)$ and $U_x(q)$ that are determined by the least solution of the following set of inequalities: for each transition (q, g, R, q') in the automaton, we have:

$$\begin{cases} L_x(q) \geq c & \text{if } x \geqslant c \text{ is a constraint in } g \\ L_x(q) \geq L_x(q') & \text{if } x \notin R \end{cases} \tag{2.3}$$

Similar inequalities are written for U, now considering $x \leqslant c$. It has been shown in [BBFL03] that such an assignment of constants is sound and complete for state reachability.

Consider again the automaton $\mathcal{A}_{stat}$ of Figure 2.14. By applying the above constraints, one would get $L_x(q_0) = U_x(q_0) = 1$, and $L_y(q_0) = U_y(q_0) = -\infty$. It is only in q_2 that we have $L_x(q_2) = U_x(q_2) = 10^6$ and $L_y(q_2) = U_y(q_2) = 10^6$. With these set of constraints the simulation graph using $Extra^+_{LU}$ consists only of 4 nodes.

One can see that this method, that performs a static analysis on the structure of the automaton, can indeed give very big gains.

2.7 Standard algorithm

The reachability problem for a timed automaton $\mathcal{A}$ can be reduced to the problem of finding a path to a final state in a simulation graph. As $Extra^+_{LU}$ is the coarsest convex abstraction known, it is the abstraction used in current implementations. Let us denote by $SG^{LU}(\mathcal{A})$ the simulation graph that uses the convex $Extra^+_{LU}$ abstraction. Each state of the automaton is associated with LU-bounds obtained by static analysis of the previous section, and the abstraction of a node (q, Z) depends on the LU-bounds of state q.

The standard algorithm to solve the reachability problem does a forward exploration of the simulation graph $SG^{LU}(\mathcal{A})$ in some search order (breadth-first search, depth-first search and the like). Starting from the initial node $(q_0, Extra^+_{LU}(Z_0))$ it computes the successor nodes on-the-fly during the search. When a new node $(q', Extra^+_{LU}(Z'))$ is obtained, it is checked if q' is final. If yes, the algorithm reports "Yes". Otherwise, it is verified if there exists an already visited node $(q', Extra^+_{LU}(Z''))$ such that

Algorithm 2.1: Reachability algorithm using $Extra^+_{LU}$ abstraction and computing bounds by static analysis [Bou04, BBFL03]

```
1   Let LU_i denote LU−bounds at state q_i
2
3   function main()
4
5   Waiting := ∅;
6   Visited := ∅;
7
8   Add (q_0, Extra+_{LU_0}(Z_0)) to Waiting
9
10  while (Waiting ≠ ∅)
11      Remove (q, Z) from Waiting;
12      if (q is accepting)
13          exit Yes
14      else if (∃ (q, Z') ∈ Visited s.t. Z ⊆ Z')
15          continue
16      else
17       for each (q_s, Extra+_{LU_s}(Z_s)) s.t. (q, Z) ⇒_{Extra+_{LU}} (q_s, Extra+_{LU_s}(Z_s)) do
18          if (Extra+_{LU_s}(Z_s) ≠ ∅)
19              Add (q_s, Extra+_{LU_s}(Z_s)) to Waiting
20
21  return No
```

$Extra^+_{LU}(Z') \subseteq Extra^+_{LU}(Z'')$. If there does exist one such node, the new node is not considered for further exploration. As $Extra^+_{LU}$ is convex, this inclusion is just an inclusion check between two zones and can be done efficiently in time $\mathcal{O}(|X|^2)$ (c.f.Table 2.1). Algorithm 2.1 gives the forward exploration procedure.

2.8 Outline

We identify two shortcomings of the standard algorithm used for the reachability problem: use of only convex abstractions, and obtaining bounds by doing a static analysis.

Non-convex abstractions

Due to concerns of efficiency, the standard algorithm uses only convex abstractions. However, there are two coarser non-convex abstractions $\mathfrak{a}_{\preccurlyeq LU}$ and $Closure_M$ already defined in literature that have been of only theoretical interest. Our first contribution is to show that one can indeed use these coarse non-convex abstractions as efficiently as the currently used convex counterparts. To make this possible, one needs an efficient inclusion test $Z \subseteq \mathfrak{a}_{\preccurlyeq LU}(Z')$ and $Z \subseteq Closure_M(Z')$. We give a quadratic algorithm for these inclusion tests in Chapter 4. Incidentally, this matches with the

complexity of checking for $Z \subseteq Z'$. The resulting inclusion test is a slight modification and can be efficiently implemented.

As a surprising bonus, we also obtain that $\mathfrak{a}_{\preccurlyeq LU}$ is the coarsest abstraction that is sound and complete for reachability among all abstractions that use only the LU-information. A similar optimality can be proved for $Closure_M$: it is the coarsest abstraction when the only available knowledge is M-information. Chapter 3 is devoted to prove this optimality.

Tightening the bounds

The static analysis approach mentioned in Section 2.6 is an optimized way of getting better LU-bounds for abstraction. Bounds are assigned to each state of the automaton. For a state q, the bounds are obtained by looking at the paths of the automaton starting from q. However, a static analysis does not know if some path of the automaton is indeed possible. In particular, static analysis considers constants from unreachable parts of the automaton.

We will see in Chapter 5 how one could use the semantic information to assign better bounds. Essentially, our aim is to assign LU-bounds for each node in the zone graph. We propose to obtain the bounds on-the-fly during the process of forward exploration. To achieve this, we give an algorithm for propagating constants during the course of exploration.

We have implemented our algorithm in a prototype tool and have checked the results on standard benchmarks in the literature. We report on some promising results in Chapter 5.

Chapter 3

Non-convex abstractions

The state space of timed automata is infinite. We have seen in the previous chapter that the standard solution to this problem is to use abstractions. The abstract semantics of a timed automaton, also called its simulation graph (Definition 2.4.2), is a finite graph that is sound and complete for reachability. A simulation graph is constructed using an abstraction operator. The coarser the abstraction, the smaller is the simulation graph obtained. However, due to implementation concerns, the state-of-the-art algorithms use only convex abstractions.

We do not know how to represent non-convex abstractions efficiently. We start this chapter by giving a framework in which we can use non-convex abstractions without having to represent them explicitly. We will see that to be able to a use a non-convex abstraction $\mathfrak{a}$, one would need an efficient inclusion test of the form $Z \subseteq \mathfrak{a}(Z')$.

We have seen in Section 2.5 that abstractions using LU-bounds are in general coarser than those using M-bounds. We have also seen that $\mathfrak{a}_{\preccurlyeq LU}$ abstraction is the coarsest known abstraction (Figure 2.12) which however has not been implemented in tools because of its non-convexity. Before considering how to plug in $\mathfrak{a}_{\preccurlyeq LU}$ to our framework, we ask ourselves the question: which is the coarsest abstraction using only LU-bounds that is sound and complete for reachability? We propose an abstraction operation abs_{LU} and prove that it is the biggest such (Theorem 3.2.10).

Quite surprisingly, it turns out that the $\mathfrak{a}_{\preccurlyeq LU}$ abstraction coincides with abs_{LU} for time-elapsed zones. As the reachability algorithm uses only time-elapsed zones, this is indeed the optimal abstraction for reachability analysis using only LU-bounds. To be able to implement it in reachability algorithms, the only missing piece is an efficient inclusion test $Z \subseteq \mathfrak{a}_{\preccurlyeq LU}(Z')$. This inclusion test will be the subject of the next chapter.

Organization of the chapter

There are three main goals in this chapter.

Goal 1: How can non-convex abstractions be used in implementations efficiently? We answer this question in Section 3.1. We propose an algorithm that can work with zone representations and use non-convex abstractions indirectly.

Goal 2: Now that we know how to use non-convex abstractions efficiently in implementations, we ask the question: which is the biggest LU-abstraction? In Section 3.2, we propose an abstraction abs_{LU} and prove that it is the biggest sound and complete abstraction that uses only LU-information.

Goal 3: How far is the $\mathfrak{a}_{\preccurlyeq LU}$ abstraction from this biggest abs_{LU} abstraction? We address this question in Section 3.3. We show that over time-elapsed zones, abs_{LU} and $\mathfrak{a}_{\preccurlyeq LU}$ actually coincide (Theorem 3.3.3). Since we work entirely with time-elapsed zones during the reachability analysis, we can infer that $\mathfrak{a}_{\preccurlyeq LU}$ is indeed the biggest LU-abstraction sound and complete for reachability.

Goals 2 and 3 deal with abstractions using LU-bounds. As a side effect, we get that the non-convex $Closure_M$ abstraction using M-bounds is the biggest M-abstraction. We show this in Section 3.4.

We provide some concluding remarks and a comprehensive summary of the chapter in Section 3.5.

3.1 Implementing non-convex abstractions

We propose a way to use non-convex abstractions and zone representations at the same time. Recall the definition of a simulation graph (Definition 2.4.2). The nodes of a simulation graph $SG^{\mathfrak{a}}(\mathcal{A})$ are made of pairs (q, W) where q is a state of the automaton and W is a set of valuations. We will only consider sets W of the form $\mathfrak{a}(Z)$ where Z is a *time-elapsed* zone.

Definition 3.1.1 (Timed-elapsed zone) A zone Z is said to be *time-elapsed* if it is closed under time-successors: that is $Z = \{v + \delta \mid v \in Z, \delta \in \mathbb{R}_{\geq 0}\}$.

We represent a set of the form $\mathfrak{a}(Z)$ by just Z. This way we can represent states of an abstract transition system efficiently: we need just to store a zone. In order for this to work we need to be able to compute the transition relation on this representation. We also need to know when two representations stand for the same node in the abstract system. This is summarized in the following two requirements:

Transition compatibility: for every transition $(q, \mathfrak{a}(Z)) \Rightarrow_{\mathfrak{a}} (q', W')$ and the matching transition $(q, Z) \Rightarrow (q', Z')$ we have $W' = \mathfrak{a}(Z')$.

Efficient inclusion test: for every two zones Z, Z', the test $Z \subseteq \mathfrak{a}(Z')$ is efficient.

The first condition also ensures that all the sets reachable by $\Rightarrow_{\mathfrak{a}}$ are of the form $\mathfrak{a}(Z)$ where Z is a time-elapsed zone. This is because we start with $\mathfrak{a}(Z_0)$ in the initial node that is time-elapsed and the symbolic transition $(q, Z) \Rightarrow (q', Z')$ always yields time-elapsed zones. This transition compatibility condition is also quite easy to satisfy. We show that every abstraction defined based on a time-abstract simulation (c.f. Definitions 2.4.4 and 2.4.5) is transition compatible. Assume that we are given an automaton $\mathcal{A}$.

Lemma 3.1.2 Let $\mathfrak{a}_{\preceq_{t.a.}}$ be an abstraction based on a time-abstract simulation relation and let Z be a time-elapsed zone. For every transition $(q, \mathfrak{a}_{\preceq_{t.a.}}(Z)) \Rightarrow_{\mathfrak{a}_{\preceq_{t.a.}}} (q', W')$ and the matching transition $(q, Z) \Rightarrow (q', Z')$, we have $W' = \mathfrak{a}_{\preceq_{t.a.}}(Z')$.

Proof
Let t be the transition corresponding to $\Rightarrow_{\mathfrak{a}_{\preceq_{t.a.}}}$ and $\Rightarrow$. We first prove that $W' \subseteq \mathfrak{a}_{\preceq_{t.a.}}(Z')$. Let $v' \in W'$. We will show that there exists a valuation in Z' that simulates v' with respect to $\preceq_{t.a.}$.

According to the definition of the abstract symbolic transition $(q, \mathfrak{a}_{\preceq_{t.a.}}(Z)) \Rightarrow_{\mathfrak{a}_{\preceq_{t.a.}}} (q', W')$, there is a valuation $v_1 \in \mathfrak{a}_{\preceq_{t.a.}}(Z)$ and a time elapse $\delta_1 \in \mathbb{R}_{\geq 0}$ such that:

$$\begin{aligned}(q, v_1) \rightarrow^t \rightarrow^{\delta_1} (q', v_1')\\ \text{and } v' \preceq_{t.a.} v_1'\end{aligned}$$

Firstly consider the intermediate configuration obtained after the $\rightarrow^t$ transition from (q, v_1). Call it $(q', \overline{v}_1')$. We know that $\overline{v}_1' \in W'$. This valuation $\overline{v}_1'$ can elapse a time δ_1 and become v_1'. Given that $\preceq_{t.a.}$ is a time-abstract simulation, this intermediate valuation $\overline{v}_1'$ can simulate v' too:

$$\begin{aligned}(q, v_1) \rightarrow^t (q', \overline{v}_1')\\ \text{and } v' \preceq_{t.a.} \overline{v}_1'\end{aligned} \tag{3.1}$$

Recall that $v_1 \in \mathfrak{a}_{\preceq_{t.a.}}(Z)$. Therefore, there exists a valuation $v_2 \in Z$ such that $v_1 \preceq_{t.a.} v_2$. As (q, v_1) can take the transition $\rightarrow^t$, by definition of time-abstract simulation, there exists a time elapse δ_2 such that (q, v_2) can take the transition after the time elapse δ_2:

$$\begin{aligned}(q, v_2) \rightarrow^{\delta_2} \rightarrow^t (q', \overline{v}_2')\\ \text{and } \overline{v}_1' \preceq_{t.a.} \overline{v}_2'\end{aligned} \tag{3.2}$$

From (3.1) and (3.2), we see that $v' \preceq_{t.a.} \overline{v}_2'$. Note that as Z is a time-elapsed zone and since $v_2 \in Z$, we also have $v_2 + \delta_2 \in Z$ and this in turn implies that $\overline{v}_2' \in Z'$. This shows that $v' \in \mathfrak{a}_{\preceq_{t.a.}}(Z')$ and hence $W \subseteq \mathfrak{a}_{\preceq_{t.a.}}(Z')$.

Algorithm 3.1: Reachability algorithm to incorporate non-convex abstractions $\mathfrak{a}_{\preceq_{t.a.}}$ based on time-abstract simulation

```
1   function main()
2
3   Waiting := ∅;
4   Visited := ∅;
5
6   Add (q_0, Z_0) to Waiting
7
8   while (Waiting ≠ ∅)
9       Remove (q, Z) from Waiting;
10      if (q is accepting)
11          exit Yes
12      else if ( ∃ (q, Z') ∈ Visited s.t. Z ⊆ a_{⪯t.a.}(Z') )
13          continue
14      else
15       for each (q_s, Z_s) s.t. (q, Z) ⇒ (q_s, Z_s) do
16          if (Z_s ≠ ∅)
17              Add (q_s, Z_s) to Waiting
18
19  return No
```

We will now show the converse: $\mathfrak{a}_{\preceq_{t.a.}}(Z') \subseteq W'$. Let $v \in \mathfrak{a}_{\preceq_{t.a.}}(Z')$. Then, there exists $v_1 \in Z$ and a $\delta_1 \in \mathbb{R}_{\geq 0}$ such that $(q, v_1) \rightarrow^t \rightarrow^{\delta_1} (q', v_1')$ and $v \preceq_{t.a.} v_1'$. By the property of an abstraction operator, we will have $v_1 \in \mathfrak{a}_{\preceq_{t.a.}}(Z)$ too. Now, directly by the definition of $(q, \mathfrak{a}_{\preceq_{t.a.}}(Z)) \Rightarrow^{\mathfrak{a}_{\preceq_{t.a.}}} (q', W')$, we get that $v \in W'$ and hence $\mathfrak{a}_{\preceq_{t.a.}}(Z') \subseteq W'$.

□

In particular, the schema discussed above would incorporate the $\mathfrak{a}_{\preccurlyeq_{LU}}$ abstraction which has been defined with respect to the simulation relation $\preccurlyeq_{LU}$ (c.f. Section 2.4). This gives us the following algorithm modified to incorporate non-convex abstractions. We list our new procedure in Algorithm 3.1 and explain in more detail below.

Algorithm

The standard algorithm described in Section 2.7 computes the simulation graph $SG^{\mathfrak{a}}(\mathcal{A})$ when the abstraction operator $\mathfrak{a}$ is convex. The same algorithm cannot be used for non-convex abstractions as it requires to represent non-convex sets explicitly. We can now describe the algorithm for the reachability problem that can incorporate non-convex abstractions. Refer to Algorithm 3.1.

Let $\mathfrak{a}_{\preceq_{t.a.}}$ be an abstraction operator based on a time-abstract simulation relation $\preceq_{t.a.}$. A forward exploration algorithm for solving the reachability problem that uses $\mathfrak{a}_{\preceq_{t.a.}}$ constructs the reachability tree of the zone

graph $ZG(\mathcal{A})$ instead of the simulation graph. Starting from the initial node (q_0, Z_0), the algorithm computes the successor nodes of the zone graph on-the-fly using some search order (for instance breadth-first search, depth-first search). When a new node (q', Z') is obtained, the algorithm first checks if there exists a visited node (q', Z'') with the same discrete state q' such that $Z' \subseteq \mathfrak{a}_{\preceq_{t.a.}}(Z'')$. If there does exist one such node, the newly visited node (q', Z') is not considered for further exploration. This is because the set of states reachable from Z' could be visited from Z'' itself, since for every valuation $v' \in Z'$ there exists a $v'' \in Z''$ such that $v' \preceq_{t.a.} v''$. This inclusion ensures termination if $\mathfrak{a}_{\preceq_{t.a.}}$ is finite.

3.2 Biggest LU-abstraction: abs_{LU}

In the previous section, we have seen how non-convex abstractions can be used for reachability analysis. We have also seen in Section 2.5 that in general, LU-abstractions are coarser than M-abstractions. A natural question is to know what is the coarsest LU-abstraction sound and complete for reachability testing. An LU-abstraction is parameterized by LU-bounds (Definition 2.4.6). So we ask: what is the biggest possible abstraction that uses only the LU-bound information?

Given L and U bounds, we know that the automata under consideration have guards only of the following form:

$$x \gtrdot 0,\ x \gtrdot 1\ , \ldots,\ x \gtrdot L_x$$
$$x \lessdot 0,\ x \lessdot 1\ , \ldots,\ x \lessdot U_x$$

However, we do not know the shape of the automata, in particular, the order in which the above guards appear in the paths of the automata.

A sound abstraction adds valuations v to a set W in such a way that for each possible path using the above guards that v can execute, there is a representative v' in W that can execute the same path. If this rule is not followed, there is one possible automaton with guards respecting the given LU-bounds for which this abstraction is not sound. We formalize this notion of LU-information: that is, guards using given LU-bounds and automata using only these guards.

Definition 3.2.1 (LU-guards, LU-automata) Given L, U bound functions, an *LU-guard* is a guard where lower bound guards use only constants bounded by L and upper bound guards use only constants bounded by U. An *LU-automaton* is an automaton using only LU-guards.

Our question can now be reworded.

> Given L and U bounds, what is the biggest abstraction that is sound and complete for all LU-automata?

We answer this question in four steps, as illustrated by the schema shown in Figure 3.1.

Step 1. We define a generic simulation relation $\sqsubseteq_{LU}$ (Definition 3.2.2) which is a union over all time-abstract simulation relations on LU-automata. Roughly the simulation relation says that $v \sqsubseteq_{LU} v'$ if all paths, using LU-guards, executed by v can be executed by v'. We define an abstraction abs_{LU} that is based on LU-simulation (Definition 3.2.3). The definition of LU-simulation is difficult to work with as it talks about infinite sequences of transitions.

Step 2. The next aim is to characterize this LU-simulation using a finite sequence of transitions (Definition 3.2.7). We want to come up with a sequence of LU-guards $seq(v)$ executed by a valuation v for which we can say $v \sqsubseteq_{LU} v'$ iff v' executes this characteristic sequence $seq(v)$. To achieve this, we go through an intermediate definition of what we call LU-regions (Definition 3.2.4). We define this sequence in Definition 3.2.7.

Step 3. Steps 1 and 2 have defined the necessary notions. We now observe that the following are equivalent (Proposition 3.2.8, Corollary 3.2.9):

- $v \sqsubseteq_{LU} v'$,
- v' can elapse some time and reach the LU-region of v
- v' can execute $seq(v)$.

Step 4. The previous step gives a finite characterization of the generic LU-simulation $\sqsubseteq_{LU}$. We use this to prove that every sound abstraction should be contained in abs_{LU}, in other words abs_{LU} is the biggest abstraction sound and complete for all LU-automata (Theorem 3.2.10).

Section 3.2.1 handles Step 1; Section 3.2.2 defines the LU-regions as mentioned in Step 2; Sections 3.2.3 and 3.2.4 handle Steps 3 and 4 respectively.

3.2.1 LU-simulation

Using LU-bounds we define a simulation relation on valuations without referring to any particular automaton; or to put it differently, by considering all LU-automata at the same time.

Definition 3.2.2 (LU-simulation) Let L, U be two functions giving an integer bound for every clock. The *LU-simulation relation* between valuations is the biggest relation $\sqsubseteq_{LU}$ such that if $v \sqsubseteq_{LU} v'$ then for every LU-guard g, and set of clocks $R \subseteq X$ we have

- if $v \xrightarrow{g,R} v_1$ for some v_1 then $v' \xrightarrow{g,R} v'_1$ for v'_1 such that $v_1 \sqsubseteq_{LU} v'_1$.

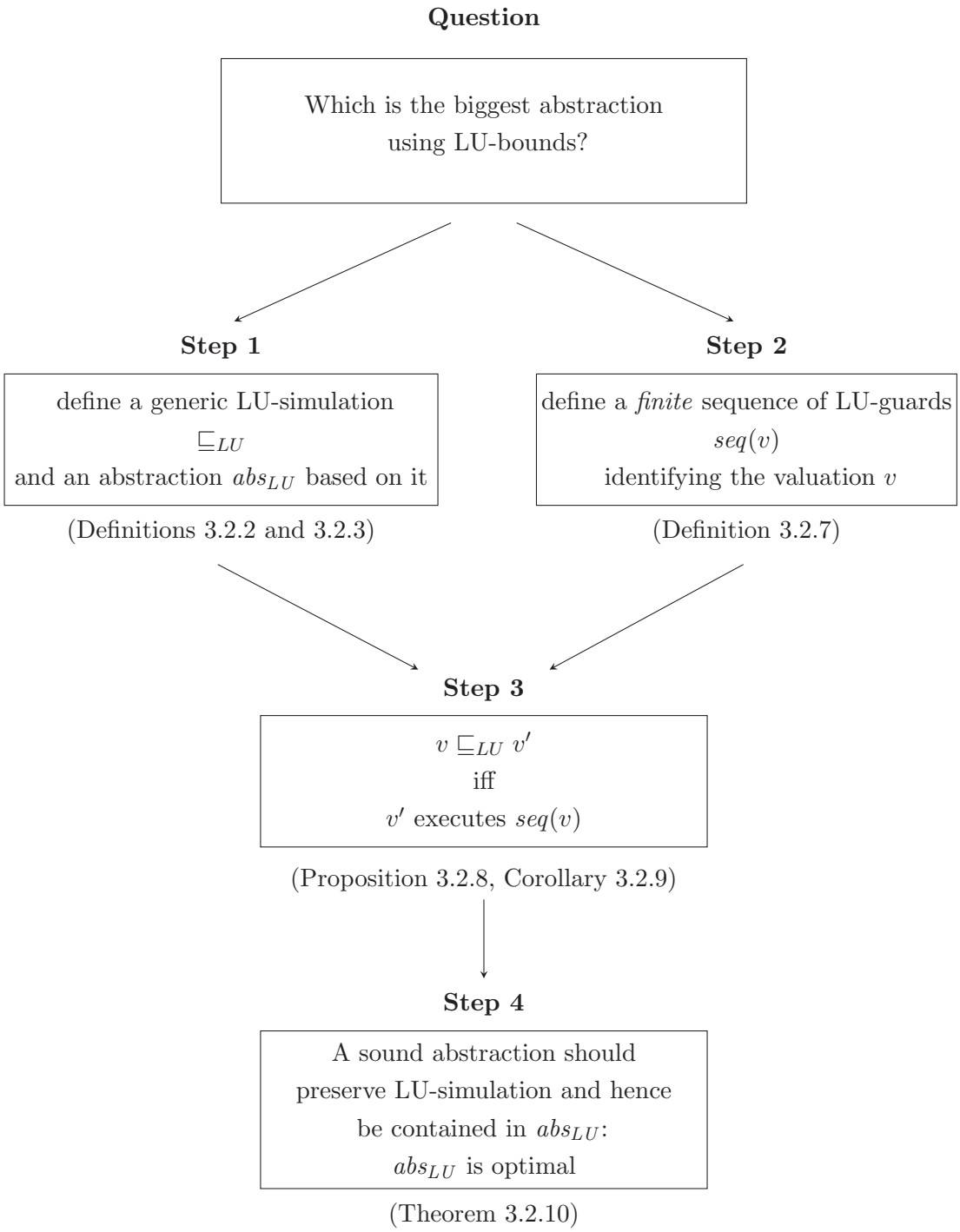


Figure 3.1: Steps to the biggest abstraction

where $v \xrightarrow{g,R} v_1$ means that for some $\delta \in \mathbb{R}_{\geq 0}$ we have $v + \delta \vDash g$ and $v_1 = [R](v + \delta)$.

Note that in the above definition, the time elapse δ' required for v' to satisfy the guard g could be different from the time elapse δ required for v to satisfy the guard g. One can check that $\sqsubseteq_{LU}$ is the biggest relation that is a time-abstract simulation for all automata with given LU bounds. We define abstraction operator abs_{LU} to be the abstraction based on this LU-simulation.

Definition 3.2.3 (Abstraction based on LU-simulation) For a zone Z we define: $abs_{LU}(Z) = \{v \mid \exists v' \in Z.\ v \sqsubseteq_{LU} v'\}$.

The definition of LU-simulation is sometimes difficult to work with since it talks about infinite sequences of actions. We will present a useful characterization implying that actually we need to consider only very particular sequences of transitions that are of length bounded by the number of clocks (Corollary 3.2.9). Essentially, we are interested in the following question: given a valuation v, when does a valuation v' LU-simulate it, that is, when is $v \sqsubseteq_{LU} v'$. We start with a preparatory definition of what we call LU-regions.

3.2.2 LU-regions

We introduce the notion of LU-regions. The classical notion of regions [AD94] (recalled in Section 2.2) depends on the maximum bounds function M. Given only the maximum bounds M, we know that there could be guards $x \lessdot c$ and $x \gtrdot c$ for $c \in \{0, \ldots, M_x\}$ in the automaton. However, with the LU-bounds, there is more information and consequently fewer guards: $x \lessdot c$ for $c \in \{0, \ldots, U_x\}$, and $x \gtrdot c$ for $c \in \{0, \ldots, L_x\}$. Note that for each x, we have $M_x = \max(L_x, U_x)$.

In the classic case, a valuation v' belongs to the region $[v]^M$ if two properties are satisfied:

Invariance by guards: v' satisfies the same guards as v,

Invariance by time-elapse: for every time elapse $\delta \in \mathbb{R}_{\geq 0}$, there is a $\delta' \in \mathbb{R}_{\geq 0}$ such that $v' + \delta' \in [v + \delta]^M$.

We would like to define a notion of LU-regions in the same spirit, now with the additional information on the guards. For this discussion let us fix some L and U functions.

Definition 3.2.4 For a valuation v we define its *LU-region*, denoted $\langle v \rangle^{LU}$, to be the set of valuations v' such that:

- v' satisfies the same LU-guards as v.

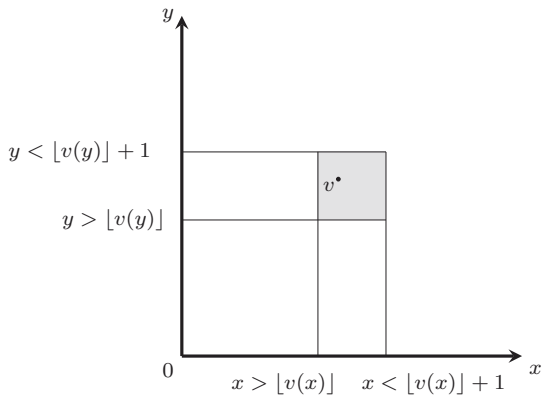


Figure 3.2: Less than both the bounds

- For every pair of clocks x, y with $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$, $\lfloor v(y) \rfloor = \lfloor v'(y) \rfloor$, $v(x) \leq U_x$ and $v(y) \leq L_y$ we have:
  - if $0 < \{v(x)\} < \{v(y)\}$ then $\{v'(x)\} < \{v'(y)\}$.
  - if $0 < \{v(x)\} = \{v(y)\}$ then $\{v'(x)\} \leq \{v'(y)\}$.

We take a closer look at the definition and see how it satisfies the two invariance properties. The first invariance with respect to guards has been directly incorporated in the first condition of the definition. We provide some insights by looking at the case with two clocks.

- Consider a valuation v and the unit square around it as shown in Figure 3.2. If $v(x) \leq \min(L_x, U_x)$ and $v(y) \leq \min(L_y, U_y)$ then all the four constraints shown in Figure 3.2 are valid LU-guards. Therefore, in this case, the valuations satisfying the first condition of Definition 3.2.4 belong to the unit square shown.

- Consider Figure 3.3. If $v(x) > L_x$, then the constraint $x > \lfloor v(x) \rfloor$ is no longer a valid LU-guard, unless $\lfloor v(x) \rfloor = L_x$. In fact, the best lower bound that one could have is $x > L_x$. So all the valuations in the shaded portion of Figure 3.3 satisfy the first condition of Definition 3.2.4.

- Dually, when $v(x) > U_x$, as shown in Figure 3.4, the upper bound constraint $x < \lfloor v(x) \rfloor + 1$ is no longer valid. There is no upper bound guard that could constrain valuations with x-coordinate greater than $v(x)$. The set of valuations satisfying the same LU-guards as v extend till infinity on the "right", as shown in the figure.

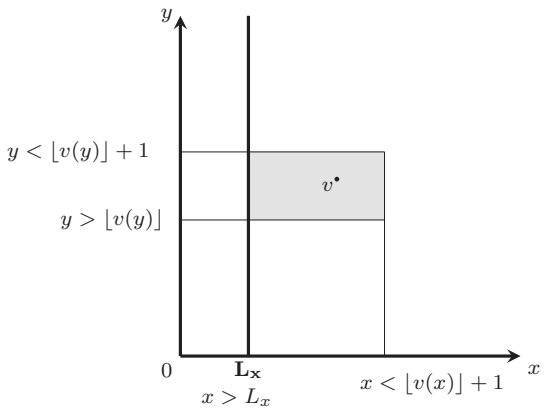


Figure 3.3: $v(x) > L_x$

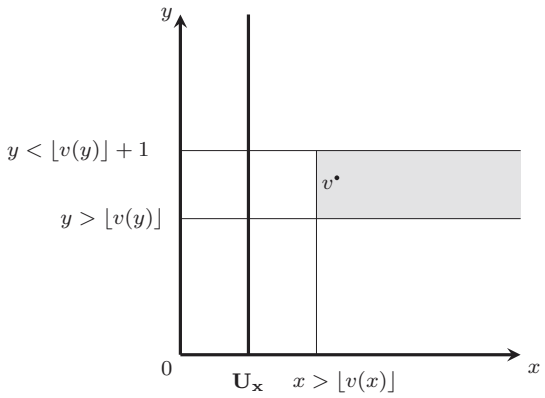


Figure 3.4: $v(x) > U_x$

As all our guards are diagonal-free, the valuations determined by the first condition of Definition 3.2.4 are bounded by either horizontal or vertical lines as shown in the above illustrations. In particular, there are no diagonals. With this condition, we are able to satisfy the property concerning the invariance by guards given in Page 48.

Let us now consider the second property which talks about invariance by time-elapse. It is for this property that we need the second condition in the definition of LU-regions. In the case of the classic regions, to satisfy this property, we required that the order between non-zero fractional parts in v and v' match for all clocks less than the M-bounds (c.f condition 4 of Definition 2.2.1). In the current case of LU, it is a bit more complicated. We motivate by an illustration.

Consider again a valuation v and the unit square around it as in Fig-

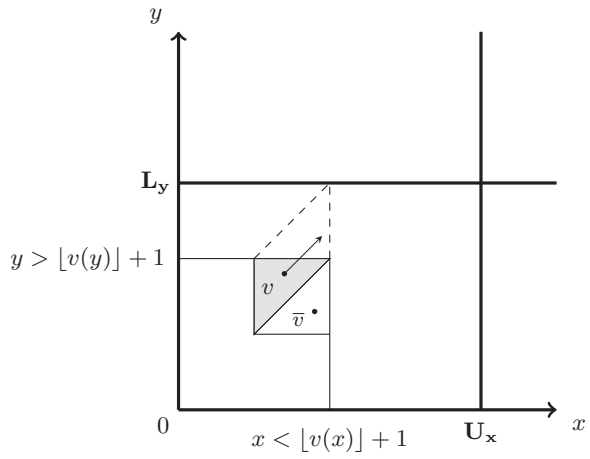


Figure 3.5: Distinguishing diagonals. The constraints $x < \lfloor v(x) \rfloor + 1$ and $x > \lfloor v(y) \rfloor + 1$ determine an area which includes the dotted triangle, where v can reach after a time elapse but $\overline{v}$ cannot.

ure 3.5. Additionally we show a valuation $\overline{v}$ which is in the same unit square, and hence satisfies the same LU-guards as v, but differs in the ordering of the fractional parts: we have $0 < \{v(x)\} < \{v(y)\}$, whereas $0 < \{\overline{v}(y)\} < \{\overline{v}(x)\}$. Look at the dotted triangle. Valuation v can let some time elapse and reach this dotted triangle. However, no matter what the time-elapse is, valuation $\overline{v}$ can never reach this triangle. If one could describe the area containing this dotted triangle by valid LU-guards, then we need to distinguish between valuations v and $\overline{v}$ in order to maintain the property of invariance by time-elapse (page 48).

Observe that if we can write $x < \lfloor v(x) \rfloor + 1$ and $y > \lfloor v(y) \rfloor + 1$, it describes an area containing the dotted triangle, and additionally $\overline{v}$ can reach nowhere in this area, for any amount of time-elapse. Therefore if $v(x) \leq U_x$ and $v(y) \leq L_y$, we require the ordering of fractional parts to match for valuations in the unit square containing v. This explains the second condition in the Definition 3.2.4 of LU-regions, which additionally makes a subtle difference between the strictly less than relation and the equality relation.

We show what happens if $v(x) > U_x$ or if $v(y) > L_y$ respectively in Figures 3.6 and 3.7. In both cases the LU-region overshoots the diagonal and adds the valuation $\overline{v}$. The only case when the diagonal is indeed maintained is when $v(x) \leq U_x$ and $v(y) \leq L_y$. Figure 3.8 gives some examples of LU-regions.

Remark 3.2.5 If $L_x = U_x = M_x$, for some bound function M and all clocks x, then we get just the usual definition of regions with respect to the set of

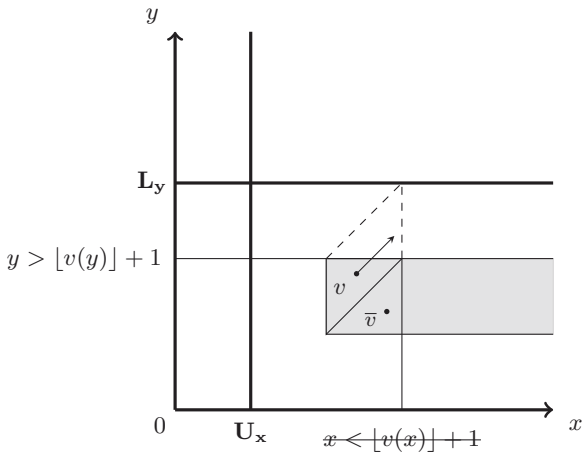


Figure 3.6: Area shown by the dotted triangle cannot be described by LU-guards: when $v(x) > U_x$ the constraint $x < \lfloor v(x) \rfloor + 1$ is not valid anymore. Diagonal overshot and $\overline{v}$ is added to $\langle v \rangle^{LU}$ shown by the shaded portion

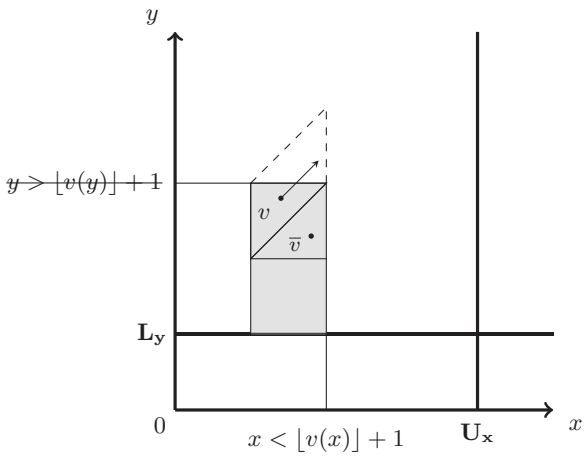


Figure 3.7: Area shown by the dotted triangle cannot be described by LU-guards: when $v(y) > L_y$ the constraint $y > \lfloor v(y) \rfloor + 1$ is not valid anymore. Diagonal overshot and $\overline{v}$ is added to $\langle v \rangle^{LU}$ shown by the shaded portion

bounds defined by M (cf. Section 2.2).

We said that the second condition in the definition of LU-regions has been added in order to obtain the invariance by time-elapse property mentioned in page 48. We saw an example over two clocks for which it was good enough. But, is it sufficient? The following lemma will now show that with the two conditions specified in the definition, one can achieve the invariance with respect to time-elapse.

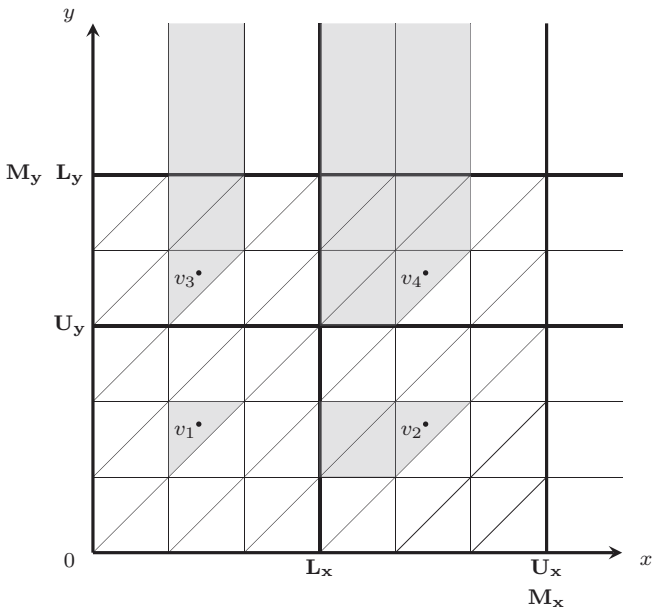


Figure 3.8: Examples of LU-regions

Lemma 3.2.6 Let v, v' be valuations such that $v' \in \langle v \rangle^{LU}$. For all $\delta \in \mathbb{R}_{\geq 0}$, there exists a $\delta' \in \mathbb{R}_{\geq 0}$ such that $v' + \delta' \in \langle v + \delta \rangle^{LU}$.

Proof
We are given valuations v and v' such that $v' \in \langle v \rangle^{LU}$. Therefore, v' satisfies the same LU-guards as v, and the following property is true for the ordering of fractional parts (this is the second condition in Definition 3.2.4 which we restate here for convenience):

$$\begin{aligned} &\text{for all } x, y \text{ with } \lfloor v(x) \rfloor = \lfloor v'(x) \rfloor,\ \lfloor v(y) \rfloor = \lfloor v'(y) \rfloor \\ &\qquad\qquad v(x) \leq U_x \text{ and } v(y) \leq L_y \\ &\text{if } 0 < \{v(x)\} < \{v(y)\} \text{ then } \{v'(x)\} < \{v'(y)\} \\ &\text{if } 0 < \{v(x)\} = \{v(y)\} \text{ then } \{v'(x)\} \leq \{v'(y)\} \end{aligned} \tag{3.3}$$

Additionally, we are given a time elapse $\delta \in \mathbb{R}_{\geq 0}$ from the valuation v. We need to construct a value $\delta' \in \mathbb{R}_{\geq 0}$ such that $v' + \delta' \in \langle v + \delta \rangle^{LU}$.

Assume $\delta < 1$. Without loss of generality, we can assume that $\delta < 1$. If $\delta \geq 1$, then we can put $\lfloor \delta' \rfloor = \lfloor \delta \rfloor$ and consider the valuations $v + \lfloor \delta \rfloor$ and $v' + \lfloor \delta' \rfloor$. As we are not altering the fractional parts in these valuations, (3.3) is true for $v + \lfloor \delta \rfloor$ and $v' + \lfloor \delta' \rfloor$. It is also easy to see that as v' satisfies the same LU-guards as v, the valuation $v' + \lfloor \delta' \rfloor$ satisfies the same LU-guards as $v + \lfloor \delta \rfloor$. This gives us $v' + \lfloor \delta' \rfloor \in \langle v + \lfloor \delta \rfloor \rangle^{LU}$ and we need to consider the

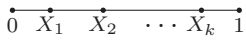
time elapse $\{\delta\}$ from $v + \lfloor\delta\rfloor$. Therefore, in the rest of the proof, without loss of generality, we can assume that $\delta < 1$.

Assume $\lfloor v(z)\rfloor = \lfloor v'(z)\rfloor$ for all clocks z. Suppose for a clock z, we have $\lfloor v(z)\rfloor < \lfloor v'(z)\rfloor$. Then, as v' satisfies the same LU-guards as v, it should be the case that $v'(z) > L_z$. So irrespective of what we choose for δ', the LU-guards with respect to z will be satisfied. Also, z does not concern the second condition of LU-regions (3.3) at all. Therefore, we can safely ignore z. Similarly, if $\lfloor v(z)\rfloor > \lfloor v'(z)\rfloor$, we know that $\lfloor v'(z)\rfloor > U_z$ and we can safely ignore this clock. In the rest of the proof, without loss of generality, we assume that $\lfloor v(z)\rfloor = \lfloor v'(z)\rfloor$ for all clocks z.

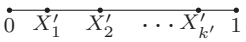
Assume $\lfloor v(z)\rfloor \leq \max(L_z, U_z)$ for all clocks z. For a clock z such that $\lfloor v(z)\rfloor$ is greater than both L_z and U_z, we know that $v'(z)$ should be greater than L_z in order to satisfy the same LU-guards. Hence any amount of time elapse would maintain this property and additionally such clocks do not concern (3.3). Hence, we assume without loss of generality that all clocks are less than at least one of the bounds.

Constructing δ'. We now have v and v' such that for all clocks z, the integral parts match, that is $\lfloor v(z)\rfloor = \lfloor v'(z)\rfloor$ and $\lfloor v(z)\rfloor \leq \max(L_z, U_z)$. Moreover, the time elapse $\delta < 1$.

Let $0 < \lambda_1 < \lambda_2 < \cdots < \lambda_k < 1$ be the fractional parts of clocks in v. Let us denote by X_i the set of clocks z that have have $\{v(z)\} = \lambda_i$. Similarly, let $0 < \lambda'_1 < \lambda'_2 < \cdots < \lambda'_{k'} < 1$ denote the fractional parts in v' and we define the set X'_i to be the set of clocks z such that $\{v'(z)\} = \lambda'_i$. This is pictorially illustrated below.

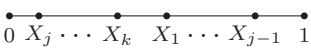


In v



In v'

After a time elapse δ from v, some of the clocks cross the next integer, whereas some of them do not. Let us say that clocks in $X_j \cup \cdots \cup X_k$ have crossed the integer. Now the fractional parts of these clocks would be smaller than those of $X_1 \cup \cdots \cup X_{j-1}$ as shown below:



In $v + \delta$

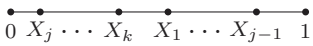
We need to choose a value δ' so that for all clocks $y \in X_j \cup \cdots \cup X_k$ such that $v + \delta(y) \leq L_y$, the time elapse δ' takes v' to the next integer.

Correspondingly for all the clocks $x \in X_1 \cup \cdots \cup X_{j-1}$ such that $v+\delta(x) \leq U_x$, the time elapse δ' still keeps v' within the same integer. Clearly we need this property to be satisfied so that $v' + \delta'$ satisfies the same LU-guards as $v + \delta$. To this regard, we define the following two values:

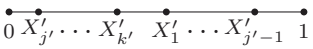
$$l = \min\{\ \{v'(y)\} \mid v + \delta(y) \leq L_y \text{ and } y \in X_j \cup \cdots \cup X_k\}$$
$$u = \max\{\ \{v'(x)\} \mid v + \delta(x) \leq U_x \text{ and } x \in X_1 \cup \cdots \cup X_{j-1}\}$$

Firstly note that $u < l$. If not, there exist clocks y and x such that $v(y) \leq L_y$ and $v(x) \leq U_x$, for which the order property given by (3.3) is not true, thus giving a contradiction. Let $\overline{\delta}$ be a value between u and l. Set $\delta' = 1 - \overline{\delta}$. By construction, $v' + \delta'$ satisfies the same LU-guards as $v + \delta$.

We will now see that this choice of δ' also satisfies (3.3) for $v' + \delta'$ and $v + \delta$. Due to δ' some clocks in v' would have crossed the next integer. Let us say that clocks in $X_{j'} \cup \cdots \cup X_{k'}$ have crossed and the others stay within the same integer. We pictorially depict the scenario with the two valuations $v + \delta$ and $v' + \delta'$ below.



In $v + \delta$



In $v' + \delta'$

Pick two clocks x, y such that:

$$\lfloor v + \delta(x) \rfloor = \lfloor v' + \delta'(x) \rfloor \text{ and } \lfloor v + \delta(y) \rfloor = \lfloor v' + \delta'(y) \rfloor \tag{3.4}$$
$$v + \delta(x) \leq U_x \text{ and } v + \delta(y) \leq L_y$$
$$\{v + \delta(x)\} < \{v + \delta(y)\} \tag{3.5}$$

Consider the case when both $x, y \in X_1 \cup \cdots \cup X_{j-1}$. As they have not crossed integer in v, they should not have crossed integer in v' too because of (3.4). Therefore both $x, y \in X'_1 \cup \cdots \cup X'_{j'-1}$. We know from (3.3) that $\{v'(x)\} < \{v'(y)\}$. Clearly the time elapse of δ' has not changed this ordering for these clocks and hence $\{v' + \delta'(x)\} < \{v' + \delta'(y)\}$. We can prove similarly when both $x, y \in X_j \cup \cdots \cup X_k$.

Let us now consider the case when $y \in X_1 \cup \cdots \cup X_{j-1}$ and $x \in X_j \cup \cdots \cup X_k$. As x has crossed integer in v, it should have crossed integer in v' too by the hypothesis (3.4). Therefore $x \in X'_{j'} \cup \cdots \cup X'_{k'}$. Again by hypothesis (3.4) the clock y should not have crossed integer and hence $y \in X'_1 \cup \cdots \cup X'_{j'-1}$. Hence we get that $\{v' + \delta'(x)\} < \{v' + \delta'(y)\}$.

This way we have proved the order property (3.3) for $v + \delta$ and $v' + \delta'$ for the case of the strict inequality. The case of the equality can be handled in a similar way. □

3.2.3 Finite paths characterizing LU-simulation

The previous section took a digression to define the notion of LU-regions. Now, we are in a position to answer the question: given two valuations v, v', when is $v \sqsubseteq_{LU} v'$. This section is devoted to show the link between this question and the definition of LU-regions. For valuations v, v', we will show that $v \sqsubseteq_{LU} v'$ if and only if v' can elapse some amount of time and fall into the LU-region of v (Proposition 3.2.8). Before that, we will define a sequence of guards that succinctly describes the LU-region $\langle v \rangle^{LU}$.

Definition 3.2.7 (LU-sequence) For a valuation v, let g_{int} be the conjunction of all LU guards that v satisfies. For every pair of clocks x, y such that $v(x) \leq U_x$, $v(y) \leq L_y$, consider guards:

- if $0 < \{v(x)\} < \{v(y)\}$ then we take a guard $g_{xy} \equiv (x < \lfloor v(x) \rfloor + 1) \wedge (y > \lfloor v(y) \rfloor + 1)$.
- if $0 < \{v(x)\} = \{v(y)\}$ then we take a guard $g_{xy} \equiv (x \leq \lfloor v(x) \rfloor + 1) \wedge (y \geq \lfloor v(y) \rfloor + 1)$.

For every y with $v(y) \leq L_y$ put $g_y = \bigwedge \{g_{xy} : v(x) \leq U_x\}$. Consider all the clocks y with $v(y) \leq L_y$ and suppose that $y_1, \ldots, y_k$ is the ordering of these clocks with respect to the value of their fractional parts: $0 \leq \{v(y_1)\} \leq \cdots \leq \{v(y_k)\}$. The LU-sequence $seq(v)$ is defined to be the sequence of transitions $\xrightarrow{g_{int}} \xrightarrow{g_{y_k}} \ \ldots \ \xrightarrow{g_{y_1}}$

Proposition 3.2.8 For every two valuations v and v':

$$v \sqsubseteq_{LU} v' \quad \text{iff} \quad \text{there is } \delta' \in \mathbb{R}_{\geq 0} \text{ with } v' + \delta' \in \langle v \rangle^{LU}.$$

Proof
First let us take v and consider its LU-sequence $seq(v)$. The sequence $seq(v)$ can be performed from v (the symbol τ denotes a time elapse):

$$v \xrightarrow{g_{int}} v \xrightarrow{\tau} v + \delta_k \xrightarrow{g_{y_k}} v + \delta_k \xrightarrow{\tau} v + \delta_{k-1} \xrightarrow{g_{y_{k-1}}} \ldots \ldots \xrightarrow{\tau} v + \delta_1 \xrightarrow{g_{y_1}} v + \delta_1$$

when choosing $\delta_i = (1 - \{v(y_i)\})$ or $\delta_i = (1 - \{v(y_i)\}) + \varepsilon$ for some sufficiently small $\varepsilon > 0$; depending on whether we test for non-strict or strict inequality in g_{y_i}. Delay δ_i makes the value of y_i integer or just above integer.

If $v \sqsubseteq_{LU} v'$, then there exists a δ' such that $v' + \delta'$ can do the sequence of transitions given by $seq(v)$. The guard g_{int} ensures that $v' + \delta'$ satisfies the same LU-guards as v. Note that in particular, this entails that for every pair of clocks x, y such that $v(x) \leq U_x$, $v(y) \leq L_y$ and $\{v(x)\} > 0$, we have:

- $\lfloor v' + \delta'(x) \rfloor < \lfloor v(x) \rfloor + 1$, and

- $\lfloor v' + \delta'(y) \rfloor \geq \lfloor v(y) \rfloor$.

Following transition g_{int}, valuation $v' + \delta'$ can satisfy guards g_{y_k} to g_{y_1} by letting some time elapse:

$$v' + \delta' \xrightarrow{g_{int}} v' + \delta' \xrightarrow{\tau} v + \delta'_k \xrightarrow{g_{y_k}} v + \delta'_k \xrightarrow{\tau} v + \delta'_{k-1} \xrightarrow{g_{y_{k-1}}} \dots$$
$$\dots \xrightarrow{\tau} v' + \delta_1 \xrightarrow{g_{y_1}} v' + \delta'_1$$

Consider the clock y_i. If the integral part $\lfloor v' + \delta'(y_i) \rfloor$ is strictly greater than $\lfloor v(y_i) \rfloor$, time elapse is not necessary to cross the guard g_{y_i}. On the other hand, if $\lfloor v' + \delta'(y_i) \rfloor = \lfloor v(y_i) \rfloor$, then for the guard g_{y_i} to be crossed, δ'_i should be sufficiently large to make the value of $v' + \delta'_i(y_i)$ integer or just above integer. But at the same time, the guard g_{xy_i} is satisfied, which entails that for all x such that $v(x) \leq U_x$, $\lfloor v(x) \rfloor = \lfloor v' + \delta'(x) \rfloor$, we get:

- if $0 < \{v(x)\} < \{v(y_i)\}$, then $\{v' + \delta'(x)\} < \{v' + \delta'(y_i)\}$ and
- if $0 < \{v(x)\} = \{v(y_i)\}$, then $\{v' + \delta'(x)\} \leq \{v' + \delta'(y_i)\}$

Therefore, from the definition of LU-regions, we get that $v' + \delta' \in \langle v \rangle^{LU}$. This shows left to right implication.

For the right to left implication we show that the relation $S = \{(v, v') : v' \in \langle v \rangle^{LU}\}$ is an LU-simulation relation. For this we take any $(v, v') \in S$, any LU guard g, and any reset R such that $v \xrightarrow{g,R} v_1$. We show that $v' \xrightarrow{g,R} v'_1$ for some v'_1 with $(v_1, v'_1) \in S$. The only non-trivial part in this is to show that if $v + \delta \vDash g$ for some δ, then there exists a δ' such that $(v + \delta, v' + \delta') \in S$ and $v' + \delta' \vDash g$. But this is exactly given by Lemma 3.2.6. □

In particular the proof shows the following.

Corollary 3.2.9 For two valuations v, v':

$$v \sqsubseteq_{LU} v' \text{ iff } v' \text{ can execute the sequence } seq(v).$$

3.2.4 Proof of optimality

We are now ready to show that $abs_{LU}(Z)$ (Definition 3.2.3), the abstraction based on $\sqsubseteq_{LU}$ simulation, is the biggest sound and complete abstraction that uses solely the LU information.

Theorem 3.2.10 *The abs_{LU} abstraction is the biggest abstraction that is sound and complete for all LU-automata. It is also finite.*

Proof
Suppose that we have some other abstraction $\mathfrak{a}'$ that is not included in abs_{LU}

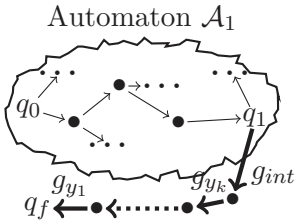


Figure 3.9: Adding the sequence $seq(v)$ to $\mathcal{A}_1$.

on at least one LU-automaton. This means that there is some LU automaton $\mathcal{A}_1$ and its reachable configuration (q_1, Z) such that $\mathfrak{a}'(Z) \setminus abs_{LU}(Z)$ is not empty. We suppose that $\mathfrak{a}'$ is complete and show that it is not sound.

Take $v \in \mathfrak{a}'(Z) \setminus abs_{LU}(Z)$. Consider the test sequence $seq(v)$ as in Corollary 3.2.9. From this corollary we know that it is possible to execute this sequence from v but it is not possible to do it from any valuation in Z since otherwise we would get $v \in abs_{LU}(Z)$.

As illustrated in Fig 3.9 we add to $\mathcal{A}_1$ a new sequence of transitions constructed from the sequence $seq(v)$. We start this sequence from q_1, and let q_f be the final state of this new sequence. The modified automaton $\mathcal{A}_1$ started in the initial configuration arrives with (q_1, Z) in q_1 and then it can try to execute the sequence we have added. From what we have observed above, it will not manage to reach q_f. On the other hand from (q_1, v) it will manage to complete the sequence. But then by completeness of the abstraction $(q_1, \mathfrak{a}'(Z)) \xrightarrow{seq(v)} (q_f, W)$ for a nonempty W. So $\mathfrak{a}'$ is not a sound abstraction.

That abs_{LU} is finite is easy to see. The set $abs_{LU}(Z)$ is a union of classical regions. Recall that we denote by M the bound function that assigns to each clock x, the maximum of L_x and U_x. Let v' be a valuation in Z. If $v' \in [v]^M$ then it is easy to see that $v' \in \langle v \rangle^{LU}$ and by definition $v' \in abs_{LU}(Z)$. □

Since abs_{LU} is the biggest abstraction, we would like to use it in a reachability algorithm. The definition of abs_{LU}, or even the characterization referring to LU-regions, are still too complicated to work with. It turns out though that there is a close link to $\mathfrak{a}_{\preccurlyeq_{LU}}$ abstraction recalled in Section 2.5.

3.3 Abstraction $\mathfrak{a}_{\preccurlyeq_{LU}}$ coincides with abs_{LU}

The $\mathfrak{a}_{\preccurlyeq_{LU}}$ abstraction proposed by Behrmann et al. in [BBLP06] has a much simpler definition. Quite surprisingly, in the context of reachability analysis the two abstractions coincide (Theorem 3.3.3). We have defined the $\mathfrak{a}_{\preccurlyeq_{LU}}$ abstraction in Section 2.5. The $\mathfrak{a}_{\preccurlyeq_{LU}}$ abstraction is based on a simulation relation $\preccurlyeq_{LU}$.

Our goal is to show that when we consider zones closed under time-

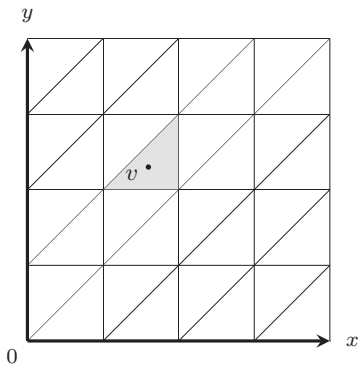


Figure 3.10: Division into neighbourhoods. The shaded portion shows $\mathsf{nbd}(v)$.

successors, $\mathfrak{a}_{\preccurlyeq_{LU}}$ and abs_{LU} coincide. To prove this, we would first show that there is a very close connection between valuations in $\langle v \rangle^{LU}$ and valuations that simulate v with respect to $\preccurlyeq_{LU}$. The following lemma says that if $v' \in \langle v \rangle^{LU}$ then by slightly adjusting the fractional parts of v' we can get a valuation v'_1 such that $v \preccurlyeq_{LU} v'_1$. We start with a preliminary definition.

Definition 3.3.1 A valuation v_1 is said to be in the *neighbourhood* of v, written $v_1 \in \mathsf{nbd}(v)$ if for all clocks x, y:

- $\lfloor v(x) \rfloor = \lfloor v_1(x) \rfloor$,
- $\{v(x)\} = 0$ iff $\{v_1(x)\} = 0$,
- $\{v(x)\} \lessdot \{v(y)\}$ implies $\{v_1(x)\} \lessdot \{v_1(y)\}$ where $\lessdot$ is either $<$ or $=$.

Notice that the neighbourhood of v is the same as the region of v with respect to the classical region definition (Section 2.2) with maximal bound being ∞. In Figure 3.10, we illustrate the division into neighbourhoods for the case of two clocks.

We give a brief intuition before proving the following lemma which gives the relation between LU-regions and $\preccurlyeq_{LU}$ simulation. Consider Figure 3.11. A valuation v is shown. Its LU-region $\langle v \rangle^{LU}$ is given by the shaded portion. Pick a valuation v' that belongs to $\langle v \rangle^{LU}$ and let us see if it satisfies $v' \preccurlyeq_{LU} v$. We recall below Definition 2.5.1 of $\preccurlyeq_{LU}$ for convenience.

►**Definition 2.5.1. LU-preorder** Let $L, U : X \to \mathbb{N}$ be two bound functions. For a pair of valuations we set $v \preccurlyeq_{LU} v'$ if for every clock x:

- if $v'(x) < v(x)$ then $v'(x) > L_x$, and
- if $v'(x) > v(x)$ then $v(x) > U_x$.

As we can see in Figure 3.11, the value of $v'(x) > v(x)$ and additionally $v'(x) \leq U_x$. This shows that $v \not\preccurlyeq_{LU} v'$ due to its x-coordinate. However,

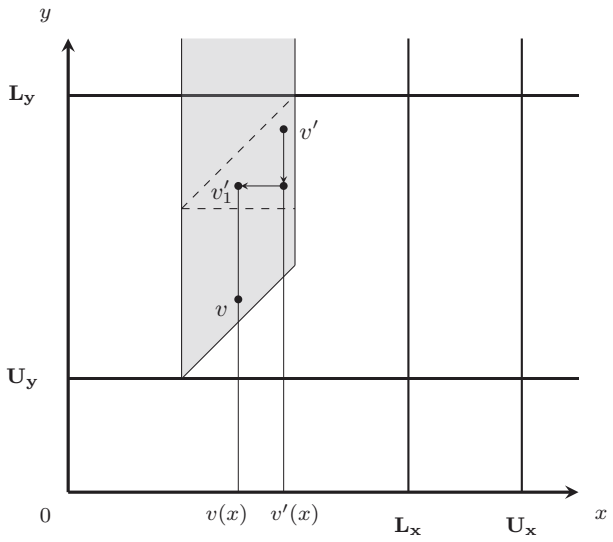


Figure 3.11: Intuition for the adjustment lemma.

by slightly adjusting the fractional parts,that is, reducing $\{v'(y)\}$ to move it down a bit and then reducing $\{v'(x)\}$ to make $v'(x)$ equal to $v(x)$ leads us to a valuation v'_1 which is in the neighbourhood of v' but now $v \preccurlyeq_{LU} v'_1$. The adjustment is depicted in Figure 3.11. Essentially, the following lemma claims that the LU-region $\langle v \rangle^{LU}$ can be obtained as the downward closure of $\preccurlyeq_{LU}$ over the set $\mathsf{nbd}(v)$, in other words, $\mathfrak{a}_{\preccurlyeq_{LU}}(\mathsf{nbd}(v))$.

Lemma 3.3.2 (Adjustment) Let v be a valuation and let $v' \in \langle v \rangle^{LU}$. Then, there exists a $v'_1 \in \mathsf{nbd}(v')$ such that $v \preccurlyeq_{LU} v'_1$.

Proof
Let $v' \in \langle v \rangle^{LU}$. The goal is to construct a valuation $v'_1 \in \mathsf{nbd}(v')$ that satisfies $v \preccurlyeq_{LU} v'_1$. To be in the neighbourhood, the valuation v'_1 should have the same integral parts as that of v' and should agree on the ordering of fractional parts. So for all x, we put $\lfloor v'_1(x) \rfloor = \lfloor v'(x) \rfloor$. It remains to choose the fractional parts for v'_1. But before, we will first see that there are clocks for which irrespective of what the fractional part is, the two conditions in Definition 2.5.1 would be true.

Consider a clock x that has $\lfloor v'(x) \rfloor < \lfloor v(x) \rfloor$. Since v' satisfies all LU-guards as v, we should have $v'(x) > L_x$. The first condition of $\preccurlyeq_{LU}$ for x becomes true and the second condition is vacuously true. Similarly, when $\lfloor v'(x) \rfloor > \lfloor v(x) \rfloor$, we should have $v(x) > U_x$ and the second condition of $\preccurlyeq_{LU}$ becomes true and the first condition is vacuously true. Therefore, clocks x that do not have the same integral part in v and v' satisfy the $\preccurlyeq_{LU}$ condition directly thanks to the different integral parts. Whatever the fractional parts of v'_1 are, the $\preccurlyeq_{LU}$ condition for these clocks would still be true.

Let us therefore now consider only the clocks that have the same integral parts: $\lfloor v'(x) \rfloor = \lfloor v(x) \rfloor$. If this integer is strictly greater than both L_x and U_x, the two conditions of $\preccurlyeq_{LU}$ would clearly be satisfied, again irrespective of the fractional parts. So we consider only the clocks x that have the same integral part in both v and v' and additionally either $\lfloor v(x) \rfloor \leq U_x$ or $\lfloor v(x) \rfloor \leq L_x$.

We prune further from among these clocks. Suppose there is such a clock that has $\{v'(x)\} = 0$. To be in the neighbourhood, we need to set $\{v'_1(x)\} = 0$. If $\{v(x)\}$ is 0 too, we are done as the $\preccurlyeq_{LU}$ condition becomes vacuously true. Otherwise, we would have $v'(x) = v'_1(x) < v(x)$. But recall that $v' \in \langle v \rangle^{LU}$ and so it satisfies the same LU-guards as v does. This entails that $v'_1(x) > L_x$ and we get the first condition of $\preccurlyeq_{LU}$ to be true. Once again, the other condition is trivial. So we eliminate clocks that have zero fractional parts in v'. A similar argument can be used to eliminate clocks that have zero fractional parts in v.

So finally, we end up with the set of clocks x that have:

- $\lfloor v'(x) \rfloor = \lfloor v(x) \rfloor$,
- $\{v'(x)\} > 0$ and $\{v(x)\} > 0$,
- $v(x) < \max(U_x, L_x)$.

Call this set X_f. The task is to select non-zero fractional values $\{v'_1(x)\}$ for all clocks x in X_f so that they match with the order in v'. This is the main challenge and this is where we would be using the second property in the definition of $v' \in \langle v \rangle^{LU}$, which we restate here:

$$\begin{aligned} &\forall x, y \in X_f \text{ such that } v(x) \leq U_x \text{ and } v(y) \leq L_y \\ &\quad 0 < \{v(x)\} < \{v(y)\} \Rightarrow \{v'(x)\} < \{v'(y)\} \\ &\quad 0 < \{v(x)\} = \{v(y)\} \Rightarrow \{v'(x)\} \leq \{v'(y)\} \end{aligned} \tag{3.6}$$

Let $0 < \lambda'_1 < \lambda'_2 < \cdots < \lambda'_n < 1$ be the fractional values taken by clocks of X_f in v', that is, for every clock $x \in X_f$, the fractional value $\{v'(x)\} = \lambda'_i$ for some $i \in \{1, \ldots, n\}$. Let X_i be the set of clocks $x \in X_f$ that have the fractional value as λ'_i:

$$X_i = \{x \in X_f \mid \{v'(x)\} = \lambda'_i\}$$

for $i \in \{1, \ldots, n\}$.

In order to match with the ordering of v', one can see that for all clocks x_i in some X_i, the value of $\{v'_1(x_i)\}$ should be the same, and if $x_j \in X_j$ with $i \neq j$, then we need to choose $\{v'_1(x_i)\}$ and $\{v'_1(x_j)\}$ depending on the order between λ'_i and λ'_j.

Therefore, we need to pick n values $0 < \sigma_1 < \sigma_2 < \cdots < \sigma_n < 1$ and assign for all $x_i \in X_i$, the fractional part $\{v'_1(x_i)\} = \sigma_i$. We show that it can be done by an induction involving n steps.

After the k^{th} step of the induction we assume the following hypothesis:

- we have picked values $0 < \sigma_{n-k+1} < \sigma_{n-k+2} < \cdots < \sigma_n < 1$,
- for all clocks $x \in X_{n-k+1} \cup X_{n-k+2} \cdots \cup X_n$, the $\preccurlyeq_{LU}$ condition is satisfied,
- for all clocks $y \in X_1 \cup X_2 \cdots \cup X_{n-k}$, we have

$$v(y) \leq L_y \Rightarrow \{v(y)\} < \sigma_{n-k+1} \tag{3.7}$$

Let us now perform the $k+1^{th}$ step and show that the induction hypothesis is true for $k+1$. The task is to pick σ_{n-k}. We first define two values $0 < l < 1$ and $0 < u < 1$ as follows:

$$l = \max\big\{\, \{v(z)\} \mid z \in X_{n-k} \text{ and } v(z) \leq L_z \,\big\}$$
$$u = \min\big\{\, \{\, \{v(z)\} \mid z \in X_{n-k} \text{ and } v(z) \leq U_z \,\} \cup \sigma_{n-k+1} \,\big\}$$

We claim that $l \leq u$. Firstly, $l < \sigma_{n-k+1}$ from the third part of the induction hypothesis. So if u is σ_{n-k+1} we are done. If not, suppose $l > u$, this means that there are clocks $x, y \in X^{n-k}$ with $v(x) \leq U_x$ and $v(y) \leq L_y$ such that $\{v(x)\} < \{v(y)\}$. From Equation 3.6, this would imply that $\{v'(x)\} < \{v'(y)\}$. But this leads to a contraction since we know they both equal λ'_{n-k} in v'.

This leaves us with two cases, either $l = u$ or $l < u$. When $l = u$, we pick $\sigma_{n-k} = l = u$. Firstly, from the third part of the hypothesis, we should have $l < \sigma_{n-k+1}$ and so $\sigma_{n-k} < \sigma_{n-k+1}$. Secondly for all $z \in X_{n-k}$, if $v'_1(z) < v(z)$, then z should not contribute to l and so $v(z) > L_z$, which is equivalent to saying, $v'_1(z) > L_z$. Similarly, if $v'_1(z) > v(z)$, then z should not contribute to u and so $v(z) > U_z$, thus satisfying the $\preccurlyeq_{LU}$ condition for z. Finally, we should show the third hypothesis. Consider a clock $y \in X_1 \cup \cdots \cup X_{n-k-1}$ with $v(y) < L_y$. If $\{v(y)\} \geq \sigma_{n-k}$, it would mean that $\{v(y)\} \geq u$ and from Equation 3.6 gives a contradiction. So the three requirements of the induction assumption are satisfied after this step in this case.

Now suppose $l < u$. Consider a clock $y \in X_1 \cup \cdots \cup X_{n-k-1}$ such that $v(y) < L_y$. From Equation 3.6, we should have $\{v(y)\} < u$. Take the maximum of $\{v(y)\}$ over all such clocks:

$$\lambda = \max\{\{v(y)\} \mid y \in X_1 \cup \cdots \cup X_{n-k-1} \text{ and } v(y) < L_y\}$$

Choose σ_{n-k} in the interval (λ, u). We can see that all the three assumptions of the induction hold after this step.

□

We are now ready to prove the second main result of this chapter. Recall that a zone is time-elapsed if it is closed with all its time-successors (Definition 3.1.1).

Theorem 3.3.3 *If Z is time-elapsed then*

$$abs_{LU}(Z) = \mathfrak{a}_{\preccurlyeq_{LU}}(Z)$$

Proof
Suppose $v \in \mathfrak{a}_{\preccurlyeq_{LU}}(Z)$. There exists a $v' \in Z$ such that $v \preccurlyeq_{LU} v'$. It can be easily verified that $\preccurlyeq_{LU}$ is a LU-simulation relation. Since $\sqsubseteq_{LU}$ is the biggest LU-simulation, we get that $v \sqsubseteq_{LU} v'$. Hence $v \in abs_{LU}(Z)$.

Suppose $v \in abs_{LU}(Z)$. There exists $v' \in Z$ such that $v \sqsubseteq_{LU} v'$. From Proposition 3.2.8, this implies there exists a δ' such that $v' + \delta' \in \langle v \rangle^{LU}$. As Z is time-elapsed, we get $v' + \delta' \in Z$. Moreover, from Lemma 3.3.2, we know that there is a valuation $v'_1 \in \mathsf{nbd}(v' + \delta')$ such that $v \preccurlyeq_{LU} v'_1$. Every valuation in the neighbourhood of $v' + \delta'$ satisfies the same constraints of the form $y - x \lessdot c$ with respect to all clocks x, y and hence v'_1 belongs to Z too. Therefore, we have a valuation $v'_1 \in Z$ such that $v \preccurlyeq_{LU} v'_1$ and hence $v \in \mathfrak{a}_{\preccurlyeq_{LU}}(Z)$. □

3.4 Biggest M-abstraction

Prior to the introduction of $\mathfrak{a}_{\preccurlyeq_{LU}}$ abstraction and the use of lower-upper bounds, it was common practice to consider only a single maximal bound function M for an automaton. In this scenario, the largest known abstraction is $Closure_M$ [Bou04]. As a side effect of Theorem 3.3.3, we get that over time-elapsed zones the $Closure_M$ abstraction is the coarsest possible.

This is because when we put $L = U = M$, the definition of LU-region $\langle v \rangle^{LU}$ becomes the classic region definition $[v]^M$ as defined in Section 2.2. So Proposition 3.2.8 now reads as: v is simulated by v' iff $v' \in [v]^M$. It is also easy to check that $v' \in [v]^M$ iff $v \in [v']^M$ too. Therefore the optimal abstraction in this case would be to close the zone with the regions intersecting it.

We define M-automata to be the set of all LU-automata with $L = U = M$. We get the following corollary to Theorem 3.3.3.

Corollary 3.4.1 When time-elapsed zones are considered, $Closure_M$ is the biggest abstraction that is sound and complete for all M-automata.

We have shown in this chapter that when time-elapsed zones are considered, abstraction $\mathfrak{a}_{\preccurlyeq_{LU}}$ is the coarsest when the only available information are the L and U bounds. For reachability analysis, as described in Section 3.1, we consider only time-elapsed zones. This shows that $\mathfrak{a}_{\preccurlyeq_{LU}}$ is the best we can do for reachability. Of course, we should not forget that such an abstraction would be of limited use if we have no efficient algorithm for handling it. This is the objective of the next chapter: we provide a quadratic algorithm to solve the inclusion $Z \subseteq \mathfrak{a}_{\preccurlyeq_{LU}}(Z')$. Once again, as a side effect, we would get an efficient inclusion test for $Z \subseteq Closure_M(Z')$.

3.5 Concluding remarks

We started the chapter with three goals in mind (c.f. page 3).

Goal 1. How can non-convex abstractions be used in implementations efficiently? We answered this question in Section 3.1 where we gave an algorithm that computes the reachability tree of the zone graph and uses inclusion tests of the form $Z \subseteq \mathfrak{a}(Z')$ for termination (Algorithm 3.1). This way, one can work with the efficient algorithms implementing zones and use the power of non-convex abstractions simultaneously.

Goal 2. Which is the biggest LU-abstraction? We answered this question in Section 3.2. We first proposed an LU-simulation relation and defined an abstraction abs_{LU} (Definitions 3.2.2 and 3.2.3). In Theorem 3.2.10, we showed that abs_{LU} is the biggest abstraction sound and complete for all LU-automata (c.f. Definition 3.2.1). The introduction to this Section 3.2 has given a break-up of the steps leading to this theorem.

Goal 3. How far is the $\mathfrak{a}_{\preccurlyeq_{LU}}$ abstraction from this biggest abs_{LU} abstraction? We showed in Theorem 3.3.3 that abstractions $\mathfrak{a}_{\preccurlyeq_{LU}}$ and abs_{LU} coincide over time-elapsed zones. As we use only time-elapsed zones in reachability analysis, we get that $\mathfrak{a}_{\preccurlyeq_{LU}}$ is the optimal abstraction for reachability. The important observation leading to this result is the Adjustment Lemma (Lemma 3.3.2) that relates the two simulations $\preccurlyeq_{LU}$ and $\sqsubseteq_{LU}$ defining $\mathfrak{a}_{\preccurlyeq_{LU}}$ and abs_{LU} respectively.

As a corollary to Theorem 3.3.3, we can get that while considering M-abstractions, $Closure_M$ abstraction is optimal for reachability analysis (Corollary 3.4.1).

The last missing piece to be able to use the $\mathfrak{a}_{\preccurlyeq_{LU}}$ abstraction is an efficient inclusion test $Z \subseteq \mathfrak{a}_{\preccurlyeq_{LU}}(Z')$. This is the subject of the next chapter.

Chapter 4

Efficient inclusion with $\mathfrak{a}_{\preccurlyeq LU}$ abstraction

In the previous chapter, we proved that over time-elapsed zones, $\mathfrak{a}_{\preccurlyeq LU}$ is the biggest abstraction that is sound and complete for all LU-automata. The objective of this chapter is to show that this biggest abstraction can efficiently be used for checking reachability.

In this chapter, we present an efficient algorithm for the inclusion $Z \subseteq \mathfrak{a}_{\preccurlyeq LU}(Z')$ (Theorem 4.5.3). Since a lot of tests of this kind need to be performed during exploration of the zone graph, it is essential to have a low complexity for this inclusion procedure. We are aiming at quadratic complexity as this is the complexity incurred in the existing algorithms for inclusions of the form $Z \subseteq Z'$ used in the standard reachability algorithm. It is well known that all the other operations needed for forward exploration, can be done in at most quadratic time (c.f. Section 2.3).

To solve the inclusion $Z \subseteq \mathfrak{a}_{\preccurlyeq LU}(Z')$, we need to check if for every valuation $v \in Z$, there is a valuation $v' \in Z'$ such that $v \preccurlyeq_{LU} v'$. See Definition 2.5.1 for the definition $\preccurlyeq_{LU}$. We will show that this reduces to asking if the LU-region (Definition 3.2.4) $\langle v \rangle^{LU}$ intersects Z'. The question of inclusion changes to a question of intersection. We show the crucial point that this intersection can be decided by verifying if the projection on every *pair* of clocks satisfies this intersection. This already gets us half way through to the main result. Once this intersection question is solved with respect to an LU-region, we extend the solution to zone Z thanks to a method allowing us to quickly tell which LU-regions intersect a given zone.

Organization of the chapter

To solve the inclusion, we consider $Z \not\subseteq \mathfrak{a}_{\preccurlyeq LU}(Z')$. The main steps to get to the inclusion are outlined in Figure 4.1.

Step 1. As a first step, we reduce the inclusion problem to a problem of

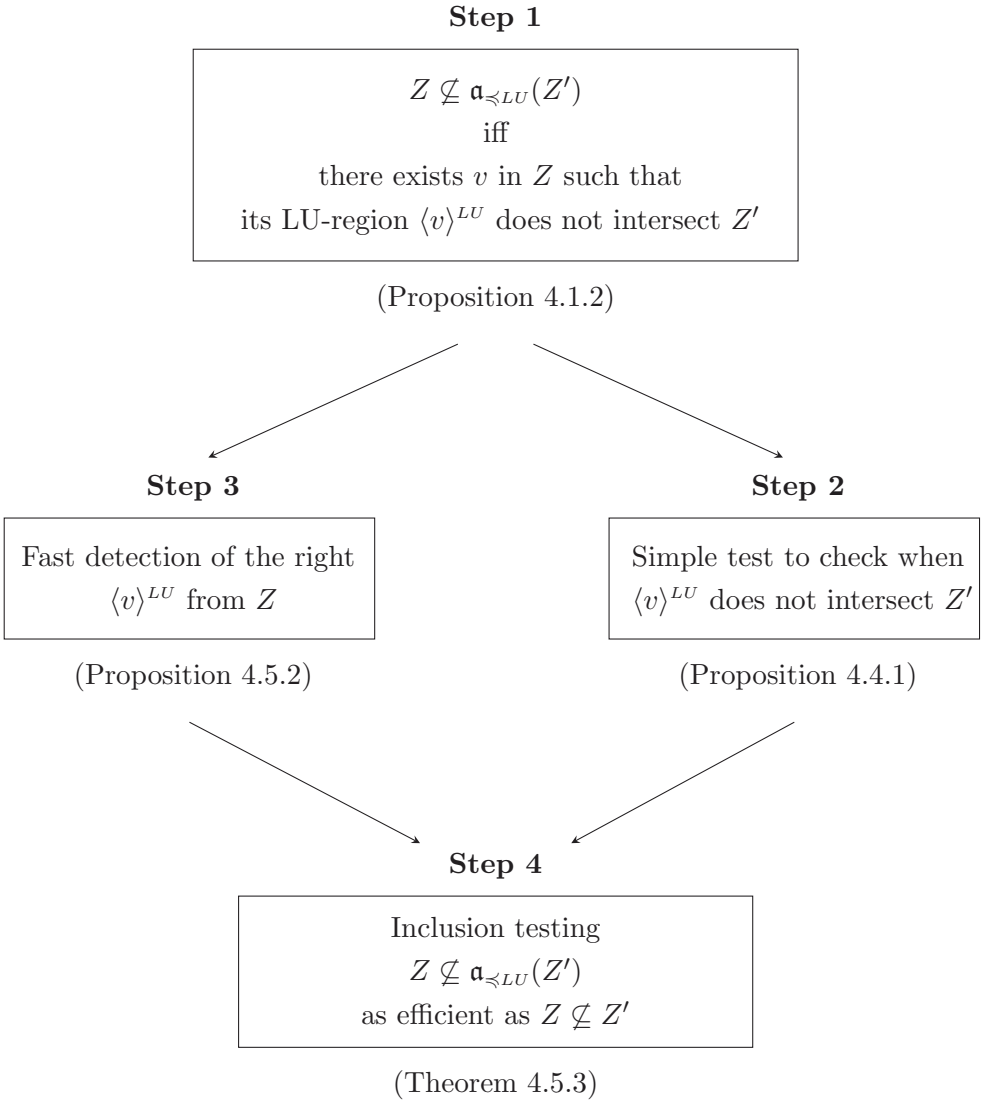


Figure 4.1: Steps to the efficient inclusion test

intersection in Section 4.1. The problem $Z \not\subseteq \mathfrak{a}_{\preccurlyeq LU}(Z')$ boils down to asking if there exists a valuation $v \in Z$ such that its LU-region $\langle v \rangle^{LU}$ does not intersect Z'.

Step 2. As a next step, we consider the intersection $\langle v \rangle^{LU} \cap Z'$. We aim to show that this intersection can be decided by looking at projections on every pair of clocks (Proposition 4.4.1). This is the most difficult step in the way to the inclusion test and spans three sections. We first describe a convenient graph representation of zones in Section 4.2. We call this the distance graph and will use it to represent Z'. Subse-

quently, in Section 4.3, we see how we can represent LU-regions as distance graphs. This gives a distance graph representation for $\langle v \rangle^{LU}$. Finally, in Section 4.4, we analyze the graph representations of $\langle v \rangle^{LU}$ and Z' to see when the intersection $\langle v \rangle^{LU} \cap Z'$ is empty.

Step 3. The previous step gives the condition for $\langle v \rangle^{LU} \cap Z'$ to be empty. We now look at the zone Z to find out quickly the valuation v that can potentially satisfy this condition.

Step 4. We substitute the valuation obtained from Step 3 to the condition of Step 2 to give the efficient test for inclusion. Both steps 3 and 4 appear in Section 4.5.

As a side effect of the inclusion test for $\mathfrak{a}_{\preccurlyeq_{LU}}$, we also get an efficient inclusion test for the $Closure_M$ abstraction by substituting $L = U = M$. We detail this inclusion test in Section 4.6.

We direct the reader to Section 4.7 (Concluding remarks) for a quick and comprehensive summary of the chapter.

For the rest of the chapter, we assume that an automaton $\mathcal{A}$ and its LU-bounds are given.

Notation

For notational convenience, we denote $v(x)$ by v_x for a valuation v and clock x.

4.1 Reducing inclusion to intersection

The aim of this chapter is to reduce the question of inclusion to a question of intersection, as depicted by the first box in Figure 4.1. The adjustment lemma (Lemma 3.3.2) shows a close connection between LU-regions and $\preccurlyeq_{LU}$-simulation in one direction: that is, if $v' \in \langle v \rangle^{LU}$ then we can find a valuation v'_1 in the neighbourhood of v' such that $v \preccurlyeq_{LU} v'_1$. We show below a connection in the other direction too.

Lemma 4.1.1 Let v, v' be valuations. If $v \preccurlyeq_{LU} v'$, then $v' \in \langle v \rangle^{LU}$.

Proof
It is not difficult to see from the definition of $\preccurlyeq_{LU}$ (Definition 2.5.1) that both v and v' satisfy the same LU-guards. It remains to show the second property for v' to be in $\langle v \rangle^{LU}$.

Let x, y be clocks such that $\lfloor v_x \rfloor = \lfloor v'_x \rfloor$, $\lfloor v_y \rfloor = \lfloor v'_y \rfloor$, and $v_x \leq U_x$, $v_y \leq L_y$. Suppose $\{v_x\} \lessdot \{v_y\}$, for $\lessdot$ being either $<$ or $=$. As $v \preccurlyeq_{LU} v'$, if $v'_x > v_x$, we need $v_x > U_x$ which is not true. Hence we can conclude that

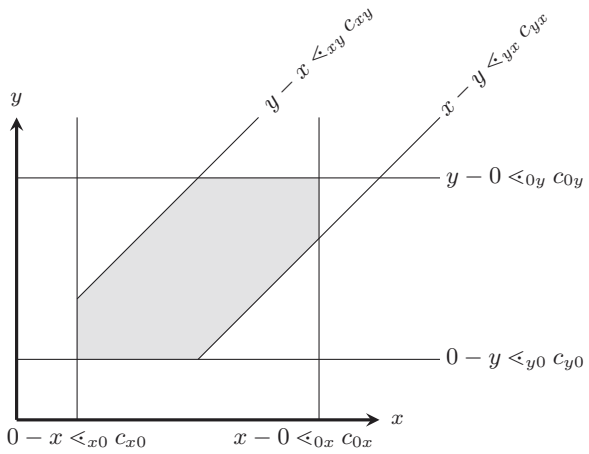


Figure 4.2: A zone with its boundaries

$v'_x \leq v_x$. Similarly, for y, one can conclude that $v'_y \geq v_y$. As the integer parts are the same in v and v', we get $\{v'_x\} < \{v'_y\}$ or $\{v'_x\} \leq \{v'_y\}$ depending on whether $\prec$ is $<$ or $=$. □

The above along with the adjustment lemma help us to reduce the question of inclusion as a question of intersection.

Proposition 4.1.2 Let Z, Z' be zones. Then, $Z \not\subseteq \mathfrak{a}_{\preccurlyeq LU}(Z')$ iff there exists a valuation $v \in Z$ such that $\langle v \rangle^{LU} \cap Z'$ is empty.

Proof
Consider the left-to-right direction. Suppose $Z \not\subseteq \mathfrak{a}_{\preccurlyeq LU}(Z')$. Then there exists a valuation $v \in Z$ such that for every valuation $v' \in Z'$ we have $v \not\preccurlyeq_{LU} v'$. Pick an arbitrary $v' \in Z'$. In particular, every valuation $v'_1 \in \mathsf{nbd}(v')$ satisfies $v \not\preccurlyeq_{LU} v'_1$. From the Adjustment lemma 3.3.2, we get that $v' \notin \langle v \rangle^{LU}$. Since v' is arbitrary, we get that $\langle v \rangle^{LU} \cap Z'$ is empty.

Now for the right-to-left direction. Suppose $\langle v \rangle^{LU} \cap Z'$ is empty. Then by Lemma 4.1.1, we get that $v \not\preccurlyeq_{LU} v'$ for every valuation $v' \in Z'$. This shows that $Z \not\subseteq \mathfrak{a}_{\preccurlyeq LU}(Z')$. □

4.2 Distance graphs

Thanks to Proposition 4.1.2, we know that to solve $Z \not\subseteq \mathfrak{a}_{\preccurlyeq LU}(Z')$, we need to check if there exists a valuation $v \in Z$ such that its LU-region $\langle v \rangle^{LU}$ does not intersect with the zone Z'. The focus now is to study this intersection, corresponding to Step 2 of Figure 4.1.

We will begin with a convenient representation of zones that we use to solve this intersection question. Consider an arbitrary zone as shown in

Figure 4.2. A zone gets defined by six constraints for every pair of variables as shown in the figure. Each constraint is a half-space. The standard way to represent a zone is to consider a DBM specifying each half space (Definition 2.3.5). Instead of considering zones as DBMs, we prefer to work with *distance graphs*, the graph representation of a zone. We begin with the description of distance graphs.

Definition 4.2.1 (Distance graph) A *distance graph* G has clocks as vertices, with an additional special vertex x_0 representing constant 0. Between every two vertices there is an edge with a *weight* of the form $(\prec, c)$ where $c \in \mathbb{Z}$ and $\prec \in \{\leq, <\}$ or $(\prec, c) = (<, \infty)$. An edge $x \xrightarrow{\prec c} y$ represents a constraint $y - x \prec c$: or in words, the distance from x to y is bounded by c. We let $[\![G]\!]$ be the set of valuations of clock variables satisfying all the constraints given by the edges of G with the restriction that the value of x_0 is 0.

For readability, we will often write 0 instead of x_0. An example of a zone with its boundaries and its distance graph representation are shown in Figure 4.3. A concrete example of a zone and its distance graph is given in Figure 4.4.

Arithmetic over weights

An arithmetic over the weights $(\prec, c)$ can be defined as follows [BY04].

Equality $(\prec_1, c_1) = (\prec_2, c_2)$ if $c_1 = c_2$ and $\prec_1 = \prec_2$.

Addition $(\prec_1, c_1) + (\prec_2, c_2) = (\prec, c_1 + c_2)$ where $\prec = <$ iff either $\prec_1$ or $\prec_2$ is $<$.

Minus $-(\prec, c) = (\prec, -c)$.

Order $(\prec_1, c_1) < (\prec_2, c_2)$ if either $c_1 < c_2$ or ($c_1 = c_2$ and $\prec_1 = <$ and $\prec_2 = \leq$).

This arithmetic lets us talk about the weight of a path as a weight of the sum of its edges.

A cycle in a distance graph G is said to be *negative* if the sum of the weights of its edges is at most $(<, 0)$; otherwise the cycle is *positive*. A distance graph is in *canonical form* if the weight of the edge from x to y is the lower bound of the weights of paths from x to y. For instance, the distance graph given in Figure 4.4 (c) is not canonical. Consider the edge from y to x. The weight of the edge is $(<, \infty)$. However there is a path $y \xrightarrow{<-2} 0 \xrightarrow{<3} x$ that has weight $(<, 1)$. The tightened edge should therefore be $y \xrightarrow{<1} x$. This corresponds to the diagonal $x - y < 1$. We depict this in Figure 4.5.

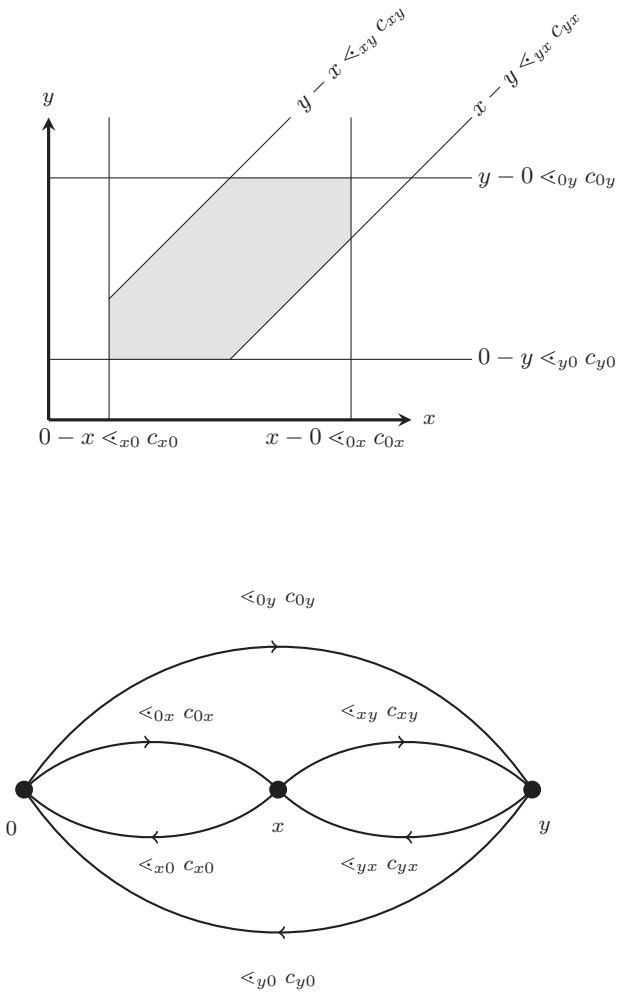


Figure 4.3: An arbitrary zone and its distance graph

Given a distance graph, its canonical form can be computed by using an all-pairs shortest paths algorithm like Floyd-Warshall's [BY04] in time $\mathcal{O}(|X|^3)$ where $|X|$ is the number of clocks. Note that the number of vertices in the distance graph is $|X| + 1$. We had listed some operations on zones in Table 2.1. We had said that the computation of $Z \wedge g$ involves a canonicalization operation. However, since g has diagonal free constraints, the canonicalization procedure involved to compute $Z \wedge g$ is easier and costs only $\mathcal{O}(|X|^2)$ [ZLZ05].

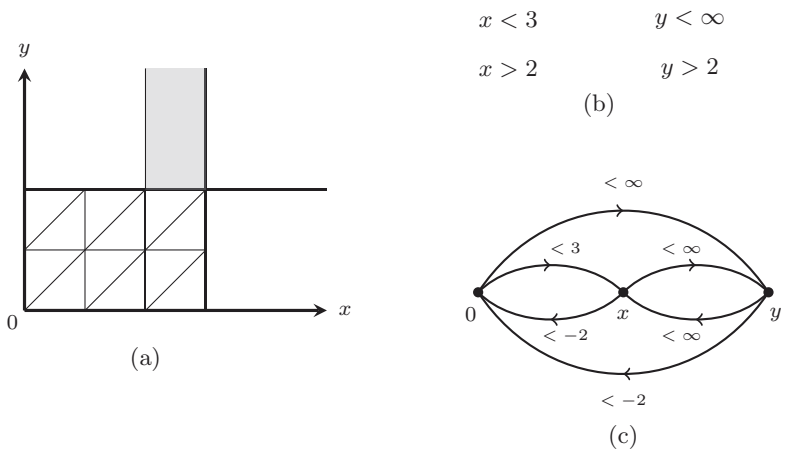


Figure 4.4: A concrete zone represented pictorially by the shaded portion in (a); by constraints defining it in (b); and by its distance graph in (c)

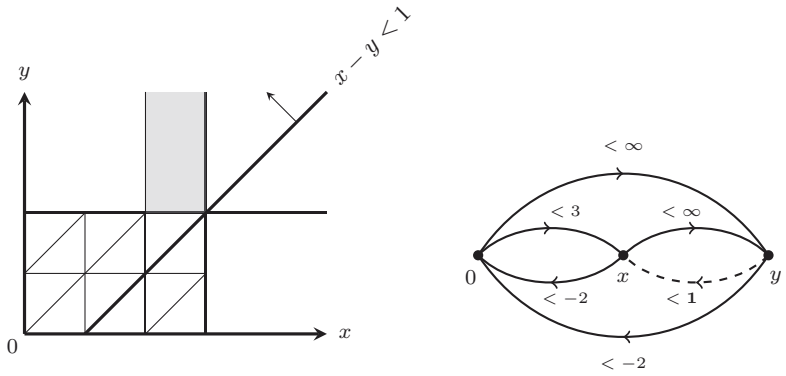


Figure 4.5: Tightening the constraints: the thick gray line in the picture on the left shows the tight diagonal bordering the shaded area, and the arrow marks the half space given by the inequality; the corresponding constraint obtained by tightening the distance graph is shown in by a dashed line in the graph.

Distance graphs with empty solution set

Recall that our aim is to prove the second box of Figure 4.1. For this, we need to know when the intersection of an LU-region and a zone is empty. In the next section we will see that an LU-region is a zone and can be represented using a distance graph. Therefore, it boils down to asking given two distance graphs G_1 and G_2 when is $[\![G_1]\!] \cap [\![G_2]\!]$ empty. Before answering this, we examine when is the solution set $[\![G]\!]$ of a distance graph G empty, in other words, when is the system of inequalities represented by the distance graph inconsistent.

Consider the distance graph of Figure 4.5. The solution set represented

by this graph is not empty. Now suppose we change the value of the edge $x \to 0$ to $(<, -4)$. This will correspond to a solution set where $x < 3$ and $x > 4$ and hence this solution set will be empty. Let us now see how this gets reflected in the distance graph. Look at the cycle $0 \xrightarrow{<3} x \xrightarrow{<-4} 0$. The sum of the weights of edges in the cycle is $(<, -1)$ which is a negative value. We will now recall the folklore result that a distance graph G has a non-empty solution set iff all cycles in G are positive.

Proposition 4.2.2 A distance graph G has only positive cycles iff $[\![G]\!] \neq \emptyset$.

Proof
If there is a valuation $v \in [\![G]\!]$ then we replace every edge $x \xrightarrow{\preccurlyeq_{xy} c_{xy}} y$ by $x \xrightarrow{\leq d} y$ where $d = v_y - v_x$. We have $d \preccurlyeq_{xy} c_{xy}$. Since every cycle in the new graph has value 0, every cycle in G is positive.

For the other direction suppose that every cycle in G is positive. Let $\overline{G}$ be the canonical form of G. Clearly $[\![G]\!] = [\![\overline{G}]\!]$, i.e., the constraints defined by G and by $\overline{G}$ are equivalent. It is also evident that all the cycles in $\overline{G}$ are positive.

We say that a variable x is *fixed* in $\overline{G}$ if in this graph we have edges $0 \xrightarrow{\leq c_x} x$ and $x \xrightarrow{\leq -c_x} 0$ for some constant c_x. These edges mean that every valuation in $[\![\overline{G}]\!]$ should assign c_x to x.

If all the variables in $\overline{G}$ are fixed then the value of every cycle in $\overline{G}$ is 0, and the valuation assigning c_x to x for every variable x is the unique valuation in $[\![\overline{G}]\!]$. Hence, $[\![\overline{G}]\!]$, and in consequence $[\![G]\!]$ are not empty.

Otherwise there is a variable, say y, that is not fixed in $\overline{G}$. We will show how to fix it. Let us multiply all the constraints in $\overline{G}$ by 2. This means that we change each arrow $x_1 \xrightarrow{\preccurlyeq c} x_2$ to $x_1 \xrightarrow{\preccurlyeq 2c} x_2$. Let us call the resulting graph H. Clearly H is in canonical form since $\overline{G}$ is. Moreover $[\![H]\!]$ is not empty iff $[\![\overline{G}]\!]$ is not empty. The gain of this transformation is that for our chosen variable y we have in H edges $0 \xrightarrow{\preccurlyeq_{0y} c_{0y}} y$ and $y \xrightarrow{\preccurlyeq_{y0} c_{y0}} 0$ with $c_{y0} + c_{0y} \geq 2$. This means that there is a natural number d such that $(\leq, d) \leq (\preccurlyeq_{0y}, c_{0y})$ and $(\leq, -d) \leq (\preccurlyeq_{y0}, c_{y0})$. Let H_d be H with edges to and from y changed to $0 \xrightarrow{\leq d} y$ and $y \xrightarrow{\leq -d} 0$, respectively. This is a distance graph where y is fixed. We need to show that there is no negative cycle in this graph.

Suppose that there is a negative cycle in H_d. Clearly it has to pass through 0 and y since there was no negative cycle in H. Suppose that it uses the edge $0 \xrightarrow{\leq d} y$, and suppose that the next used edge is $y \xrightarrow{\preccurlyeq_{yx} c_{yx}} x$. The cycle cannot come back to y before ending in 0 since then we could construct a smaller negative cycle. Hence all the other edges in the cycle come from H. Since H is in the canonical form, a path from x to 0 can be replaced by the edge from x to 0, and the value of the path will not increase. This means that our hypothetical negative cycle has the form $0 \xrightarrow{\leq d} y \xrightarrow{\preccurlyeq_{yx} c_{yx}} x \xrightarrow{\preccurlyeq_{x0} c_{x0}} 0$.

By canonicity of H we have $(\prec_{yx}, c_{yx}) + (\prec_{x0}, c_{x0}) \geq (\prec_{y0}, c_{y0})$. Putting these two facts together we get

$$(\leq, 0) > (\leq, d) + (\prec_{yx}, c_{yx}) + (\prec_{x0}, c_{x0}) \geq (\leq, d) + (\prec_{y0}, c_{y0})$$

but this contradicts the choice of d which supposed that $(\leq, d) + (\prec_{y0}, c_{y0})$ is positive. The proof when the hypothetical negative cycle passes through the edge $y \xrightarrow{\leq -d} 0$ is analogous.

Summarizing, starting from G that has no negative cycles we have constructed a graph H_d that has no negative cycles, and has one more variable fixed. We also know that if $[\![H_d]\!]$ is not empty then $[\![G]\!]$ is not empty. Repeatedly applying this construction we get a graph where all the variables are fixed and no cycle is negative. As we have seen above the semantics of such a graph is not empty. □

Intersection of two distance graphs

For two distance graphs G_1, G_2 which are not necessarily in canonical form, we denote by $\min(G_1, G_2)$ the distance graph where each edge has the weight equal to the minimum of the corresponding weights in G_1 and G_2. Even though this graph may be not in canonical form, it should be clear that it represents intersection of the two arguments, that is, $[\![\min(G_1, G_2)]\!] = [\![G_1]\!] \cap [\![G_2]\!]$; in other words, the valuations satisfying the constraints given by $\min(G_1, G_2)$ are exactly those satisfying all the constraints from G_1 as well as G_2.

From Proposition 4.2.2, the intersection $[\![G_1]\!] \cap [\![G_2]\!]$ is empty iff the distance graph $\min(G_1, G_2)$ has a negative cycle. Figure 4.6 shows an example.

4.3 LU-regions as distance graphs

Our aim is to solve Step 2 of Figure 4.1. We saw in the previous section that zones can be conveniently and uniquely represented by distance graphs. In this section, we see how we can canonically represent an LU-region of a valuation as a distance graph.

Recall the classic notion of regions and the region equivalence in Section 2.2. We repeat the constructive definition of regions below for convenience.

►Definition 2.2.3. (Region:constructive definition)
A region with respect to bound function M is the set of valuations specified as follows:

1. for each clock $x \in X$, one constraint from the set:

   $\{x = c \mid c = 0, \ldots, M_x\} \cup \{c-1 < x < c \mid c = 1, \ldots, M_x\} \cup \{x > M_x\}$

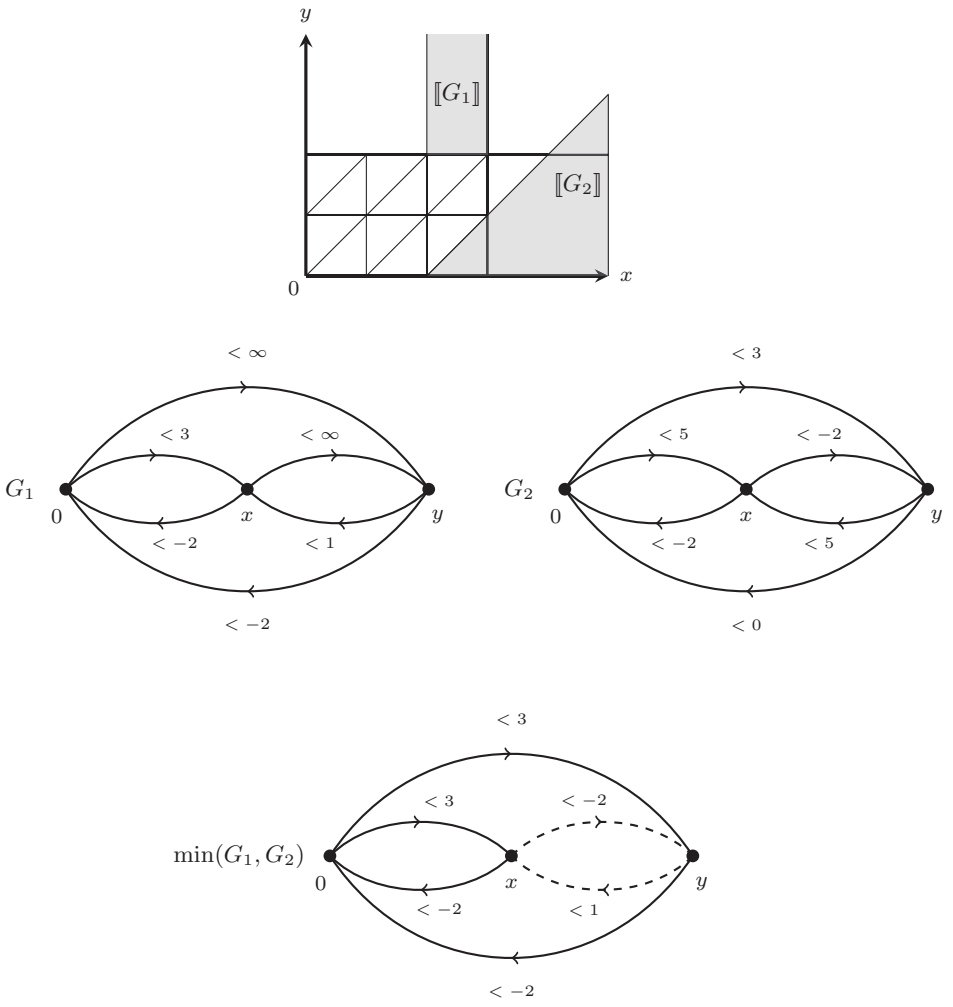


Figure 4.6: Two zones $[\![G_1]\!]$ and $[\![G_2]\!]$ are shown in the topmost figure. The corresponding distance graphs are given below. Note that $[\![G_1]\!]$ does not intersect $[\![G_2]\!]$. This is captured by the distance graph $\min(G_1, G_2)$ which has a negative cycle. Additionally, the $\min(G_1, G_2)$ graph shows exactly which constraints are responsible for non-intersection.

2. for each pair of clocks x, y having interval constraints: $c - 1 < x < c$ and $d - 1 < y < d$, it is specified if $\{x\}$ is less than, equal to or greater than $\{y\}$.

The distance graph representing a region can be constructed using the above constructive definition of a region. Consider Figure 4.4 once again. The shaded portion in (a) is in fact a region with respect to the bounds represented by the thick lines $x = 3$ and $y = 2$. Figure 4.4 (b) and (c) show the constraints and the distance graph representing the region.

For a valuation v, let G_v^M denote the canonical distance graph representing the region $[v]^M$. We are now interested in getting the LU-region of v, that is, $\langle v \rangle^{LU}$ as a distance graph. Let us recall the definition of LU-regions.

►**Definition 3.2.4. (LU-region)**
For a valuation v we define its *LU-region*, denoted $\langle v \rangle^{LU}$, to be the set of valuations v' such that:

- v' satisfies the same LU-guards as v.
- For every pair of clocks x, y with $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$, $\lfloor v(y) \rfloor = \lfloor v'(y) \rfloor$, $v(x) \leq U_x$ and $v(y) \leq L_y$ we have:
  - if $0 < \{v(x)\} < \{v(y)\}$ then $\{v'(x)\} < \{v'(y)\}$.
  - if $0 < \{v(x)\} = \{v(y)\}$ then $\{v'(x)\} \leq \{v'(y)\}$.

For a valuation v, we need to collect all valuations v' satisfying the above two conditions to get $\langle v \rangle^{LU}$. We begin with a motivating example in Figure 4.7. In the (a) part of the figure, we consider a valuation v such that $v_x > U_x$. The shaded portion in Figure 4.7 (a) shows the region $[v]^M$ and the finite valued constraints $x - y$ and $x - 0$ bounding this region (c.f. Figure 4.2 for information about boundaries). The LU-region $\langle v \rangle^{LU}$ is shown in Figure 4.7 (b). Observe that it matches with the definition given above. But more importantly, note that it can be seen as a transformation of $[v]^M$ by moving constraints $x - y$ and $x - 0$ to infinity and keeping the rest same.

We now consider a valuation v with $v_y > L_y$ in Figure 4.7 (c). Once again, the shaded portion shows the region $[v]^M$ and the interesting boundaries. We depict the LU-region $\langle v \rangle^{LU}$ in Figure 4.7 (d) matching the definition given above. Note that it can be seen as a transformation of $[v]^M$ by moving the constraint $x - y$ to infinity and the constraint $0 - y$ upto L_y. However, when we move $x - y$ to infinity, the graph that we get would no longer be canonical. We could then consider the canonicalization of the transformed graph.

Let G_v^{LU} denote the canonical distance graph representing the LU-region $\langle v \rangle^{LU}$. We will define the distance graph G_v^{LU} as a transformation of the distance graph G_v^M.

Definition 4.3.1 (Distance graph G_v^{LU}) Let v be valuation. Given the distance graph of the region $[v]^M$ in canonical form $G_v^M = (\preccurlyeq_{xy}, c_{xy})_{x,y \in X}$, first define the distance graph $G' = (\preccurlyeq'_{xy}, c'_{xy})_{x,y \in X}$ as follows:

$$(\preccurlyeq'_{yx}, c'_{yx}) = \begin{cases} (<, \infty) & \text{if } v_x > U_x \\ (<, \infty) & \text{if } v_y > L_y \text{ and } x \neq 0 \\ (<, -L_y) & \text{if } v_y > L_y \text{ and } x = 0 \\ (\preccurlyeq_{yx}, c_{yx}) & \text{otherwise} \end{cases}$$

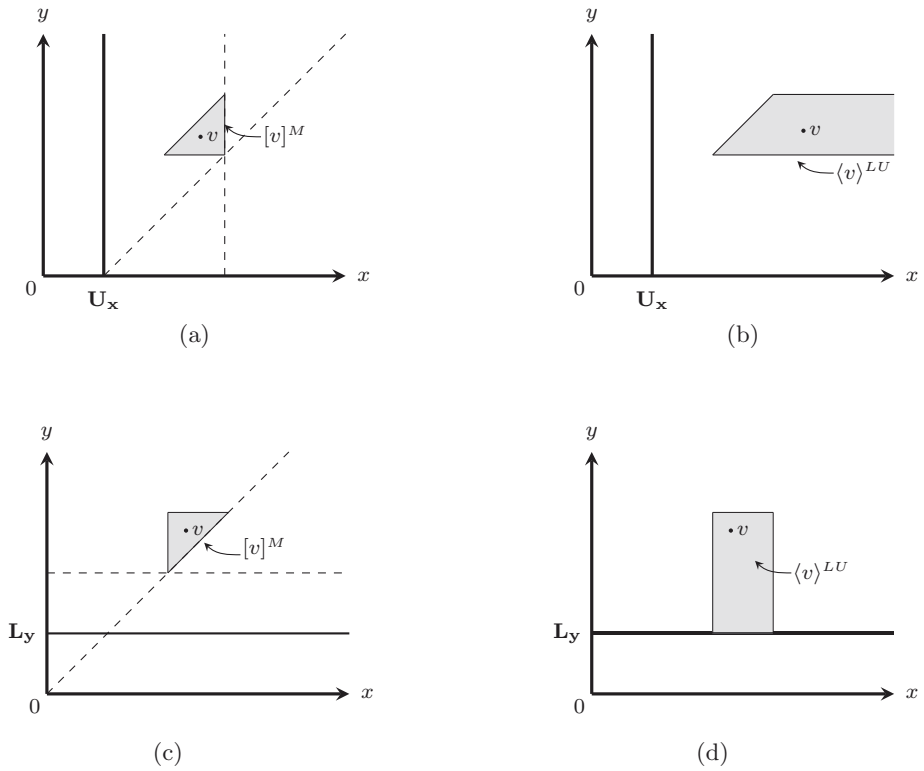


Figure 4.7: $\langle v \rangle^{LU}$ can be thought of as a transformation of $[v]^M$ by altering select constraints. Pictures (a) and (b) handle the case when $v_x > U_x$; pictures (c) and (d) handle the case when $v_y > L_y$

Then, the distance graph G_v^{LU} is defined to be the canonical form of G'.

The following lemma confirms that the distance graph defined above indeed represents $\langle v \rangle^{LU}$.

Lemma 4.3.2 Let v be a valuation. Let G_v^{LU} be the graph obtained by Definition 4.3.1. Then the sets $\langle v \rangle^{LU}$ and $[\![G_v^{LU}]\!]$ are equal.

Proof
Let $v' \in \langle v \rangle^{LU}$. We will now show that $v' \in [\![G_v^{LU}]\!]$. First observe that $[\![G_v^{LU}]\!] = [\![G']\!]$ where G' is the graph as in Definition 4.3.1. Therefore it is sufficient to show that v' satisfies the constraints given by G'. From the definition, it is clear that an edge $y \to x$ is finite valued in G' only if $v_x \leq U_x$. Additionally when $v_y \leq L_y$, the value of the edge $y \to x$ is the same as that in G_v^M. Otherwise if $v_y > L_y$, the only finite value is $(<, -L_y)$ for the edge $y \to x_0$.

Since $v' \in \langle v \rangle^{LU}$, it satisfies the same LU-guards as v. If y is a clock such that $v_y > L_y$ then $v'_y > L_y$ too. So v' satisfies constraints of the form

$y \xrightarrow{<-L_y} x_0$. It now remains to look at edges $y \xrightarrow{\preccurlyeq d} x$ with $v_y \leq L_y$, $v_x \leq U_x$ and the weight $(\preccurlyeq, d)$ coming from G_v^M. Let $\lfloor v_x \rfloor$ and $\lfloor v_y \rfloor$ be denoted as c_x and c_y respectively. As v' satisfies the same LU-guards as v, we have:

$$v'_x < c_x + 1$$
$$v'_y \geq c_y$$

Therefore $v'_x - v'_y < c_x + 1 - c_y$. Since G_v^M represents the region containing v, by definition of regions, the constant in the weight $(\preccurlyeq, d)$ is either $c_x - c_y + 1$ or $c_x - c_y$. If it is the former, then clearly, v' also satisfies this constraint. We need to consider the latter case, that is, d is $c_x - c_y$. Now if either $v'_x < c_x$ or $v'_y \geq c_y + 1$, we are done. We are left with considering the case when $\lfloor v'_x \rfloor = c_x$ and $\lfloor v'_y \rfloor = c_y$. We have:

$$\begin{aligned}
& v_x - v_y \preccurlyeq c_x - c_y \\
\Rightarrow \quad & \{v_x\} - \{v_y\} \preccurlyeq 0 \\
\Rightarrow \quad & \{v'_x\} - \{v'_y\} \preccurlyeq 0 \qquad (\text{as } v' \in \langle v \rangle^{LU}) \\
\Rightarrow \quad & \lfloor v'_x \rfloor + \{v'_x\} - (\lfloor v'_y \rfloor + \{v'_y\}) \preccurlyeq c_x - c_y \\
\Rightarrow \quad & v'_x - v'_y \preccurlyeq d
\end{aligned}$$

This proves that if $v' \in \langle v \rangle^{LU}$, then v' satisfies the constraints of G' and hence $v' \in [\![G']\!]$.

Now for the other direction, assume $v' \in [\![G']\!]$. We will show that $v' \in \langle v \rangle^{LU}$. Let x, y be clocks such that $v_x \leq U_x$ and $v_y \leq L_y$. From the definition of $[\![G']\!]$, edges of the form $y \to x_0$ and $x_0 \to x$ are retained as in G_v^M. Since $v' \in [\![G']\!]$, it is clear that v' satisfies the same LU-guards as v. We now consider the order property for LU-regions. By definition of G', the edge $y \to x$ in G' has the same weight as that in G_v^M. Further assume $\lfloor v'_x \rfloor = \lfloor v_x \rfloor = c_x$ and $\lfloor v'_y \rfloor = \lfloor v_y \rfloor = c_y$. Let the edge weight $y \to x$ be $(\preccurlyeq, d)$. We have:

$$\begin{aligned}
& v_x - v_y \preccurlyeq d \\
\Rightarrow \quad & \{v_x\} - \{v_y\} \preccurlyeq d - (c_x - c_y)
\end{aligned}$$

If $\{v_x\} < \{v_y\}$ then either $d - (c_x - c_y) < 0$ or if it is 0 then $\preccurlyeq$ is the strict inequality. As this edge remains in G', the valuation v' satisfies $v'_x - v'_y \preccurlyeq d$. Moreover, since the integral parts of v' match, we get $\{v'_x\} - \{v'_y\} \preccurlyeq d - (c_x - c_y)$. By the aforementioned property, we get $\{v'_x\} < \{v'_y\}$. A similar argument follows for the case when $\{v_x\} = \{v_y\}$. □

Before we use the distance graph G_v^{LU} for further analysis, recall that we first defined a graph G' in Definition 4.3.1 and then obtained G_v^{LU} by canonicalizing it. We will now observe some properties of G_v^{LU} that are either retained from G' or obtained thanks to canonicalization. These observations would be important in the next section when we do the analysis on the distance graph representing LU-region $\langle v \rangle^{LU}$ and zone Z'.

Lemma 4.3.3 Let v be a valuation. Let G_v^M, G_v^{LU} be the canonical distance graphs of $[v]^M$ and $\langle v\rangle^{LU}$ respectively. For variables x, y, if the edge $y \to x$ has a finite value in $\langle v\rangle^{LU}$, then:

1. $v_x \leq U_x$,
2. if $v_y \leq L_y$, the value of $y \to x$ in G_v^{LU} and G_v^M are equal,
3. if $v_y > L_y$, the value of $y \to x$ in G_v^{LU} equals the value of the path $y \to x_0 \to x$ in G_v^{LU}.

Proof
The graph G_v^{LU} is the canonical form of the graph G' defined in Definition 4.3.1. By definition, if $v_x > U_x$, all incoming edges to x in G' have weight $(<, \infty)$. So, the shortest path in this graph G' from a variable y to a variable x such that $v_x > U_x$ is $(<, \infty)$. Therefore, if in the canonical form G_v^{LU}, the edge $y \to x$ is finite valued, we should have $v_x \leq U_x$. This gives the first part of the lemma.

Consider the second part of the lemma. We know that $v_y \leq L_y$ and from the first part of the lemma, we know that $v_x \leq U_x$. The weight of $y \to x$ in G' is the same as that of G_v^M according to Definition 4.3.1. Note that the finite values in the graph G' are either the same as that of G_v^M or of the form $(<, -L_z)$ for some edges $z \to 0$. In the latter case, we also know by definition that $v_z > L_z$. Therefore the value $(<, -L_z)$ is greater than the corresponding value in G_v^M. As G_v^M is canonical, the shortest path from y to x in G' cannot reduce from its value in G_v^M and hence equals just the edge value $y \to x$. This gives the second part of the lemma.

Finally consider the third part. Assume $v_y > L_y$. From Part 1, we know that $v_x \leq U_x$. By Definition 4.3.1, the weight of $y \to x$ equals $(<, \infty)$ if x is not x_0. The only finite valued outgoing edge from y is $y \to x_0$. Therefore, we can infer two things: the shortest path from y to x_0 is given by the edge $y \to x_0$; and the shortest path from y to x should contain this edge $y \to x_0$. Secondly, note that variable x_0 has $v_{x_0} \leq L_{x_0}$ ($v_{x_0} = 0 = L_{x_0}$). By definition, the value of $x_0 \to x$ in G' is given by the corresponding value in G_v^M and by Part 2, we know that this value stays in G_v^{LU}, that is, the shortest path from x_0 to x in G' is given by the direct edge. Summing up, the shortest path from y to x in G' is given by $y \to x_0 \to x$, where both $y \to x_0$ and $x_0 \to x$ are values coming from G_v^{LU}. □

4.4 When does an LU-region intersect a zone.

We are now in a position to tackle Step 2 of Figure 4.1 and characterize the intersection $\langle v\rangle^{LU} \cap Z'$.

Recall that we concentrate on asking when is $Z \not\subseteq \mathfrak{a}_{\preccurlyeq LU}(Z')$. From Proposition 4.1.2, we know that this is true iff there is a valuation $v \in Z$

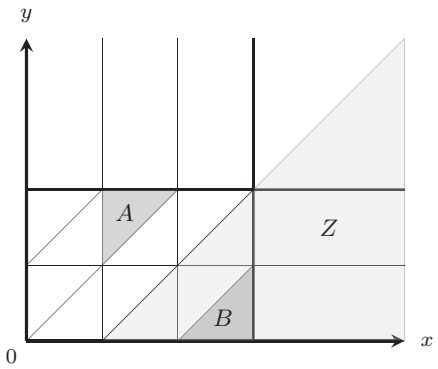


Figure 4.8: A bounded region is either totally contained in a zone Z or totally disjoint from it. It can never intersect partially. For example, the bounded region B is totally inside Z, however the bounded region A is totally disjoint from Z.

such that $\langle v \rangle^{LU}$ does not intersect Z'. Let G_v^{LU} as defined in the previous section be the canonical distance graph of $\langle v \rangle^{LU}$ and let $G_{Z'}$ be the canonical distance graph of Z'. By Proposition 4.2.2, the intersection $\langle v \rangle^{LU} \cap Z'$ is empty iff $\min(G_v^{LU}, G_{Z'})$ has a negative cycle.

We will now state a necessary and sufficient condition for the graph $\min(G_v^{LU}, G_{Z'})$ to have a negative cycle. We denote by Z'_{xy} the weight of the edge $x \to y$ in $G_{Z'}$. Similarly we denote $\langle v \rangle_{xy}^{LU}$ for the weight of $x \to y$ in G_v^{LU}. When a variable x represents the special clock x_0, we define $\langle v \rangle_{0x}^{LU}$ and $\langle v \rangle_{x0}^{LU}$ to be $(\leq, 0)$. Since by convention x_0 is always 0, this is consistent.

Proposition 4.4.1 Let v be a valuation and Z' a zone. The intersection $\langle v \rangle^{LU} \cap Z'$ is empty iff there exist two variables x, y such that $v_x \leq U_x$ and $Z'_{xy} + \langle v \rangle_{yx}^{LU} < (\leq, 0)$.

To prove the above proposition, we need a small but a crucial observation that exploits the special structure of regions. A variable x is said to be bounded in valuation v if $v_x \leq \max(L_x, U_x)$. If x, y are bounded in v, then the projection of the region $[v]^M$ onto x, y has very specific boundaries. Call it a bounded region. The following lemma makes use of the fact that a bounded region is either fully contained in a zone or is totally disjoint from it, that is, there cannot be a partial intersection of the bounded region and zone, as illustrated in Figure 4.8.

Lemma 4.4.2 Let x, y be bounded variables of v appearing in some negative cycle N of $\min(G_v^{LU}, G_{Z'})$. Let the edge weights be $x \xrightarrow{\lessdot_{xy} c_{xy}} y$ and $y \xrightarrow{\lessdot_{yx} c_{yx}} x$ in G_v^M. If the value of the path $x \to \ldots \to y$ in N is strictly less than $(\lessdot_{xy}, c_{xy})$, then $x \to \ldots \to y \xrightarrow{\lessdot_{yx} c_{yx}} x$ is a negative cycle.

Proof

Let the path $x \to \ldots \to y$ in N have weight $(\lessdot, c)$. Now, since x and y are bounded variables in v, we can have either $y - x = d$ or $d - 1 < y - x < d$ for some integer d in G_v^M.

In the first case, we have edges $x \xrightarrow{\leq d} y$ and $y \xrightarrow{\leq -d} x$ in G_v^M, that is $(\lessdot_{xy}, c_{xy}) = (\leq, d)$ and $(\lessdot_{yx}, c_{yx}) = (\leq, -d)$. Since by hypothesis $(\lessdot, c)$ is strictly less than $(\leq, d)$, we have either $c < d$ or $c = d$ and $\lessdot$ is the strict inequality. Hence $(\lessdot, c) + (\leq, -d) < (\leq, 0)$ showing that $x \to \ldots \to y \xrightarrow{\lessdot_{yx} c_{yx}} x$ is a negative cycle.

In the second case, we have edges $x \xrightarrow{<d} y$ and $y \xrightarrow{<-d+1} x$ in G_v^M, that is, $(\lessdot_{xy}, c_{xy}) = (<, d)$ and $(\lessdot_{yx}, c_{yx}) = (<, -d)$. Here $c < d$ and again $x \to \ldots \to y \xrightarrow{\lessdot_{yx} c_{yx}} x$ gives a negative cycle. □

We can now prove Proposition 4.4.1. This is the most important observation used in getting the final inclusion test.

Proof of Proposition 4.4.1

The distance graph $\min(G_v^{LU}, G_{Z'})$ represents the set $\langle v \rangle^{LU} \cap Z'$. By Proposition 4.2.2, the intersection is empty iff $\min(G_v^{LU}, G_{Z'})$ has a negative cycle. If there exist variables x, y such that $Z'_{xy} + \langle v \rangle_{yx}^{LU} < (\leq, 0)$, then there is a negative cycle $x \to y \to x$ in $\min(G_v^{LU}, G_{Z'})$ and hence $\langle v \rangle^{LU} \cap Z'$ is empty. This shows the right-to-left direction.

The left-to-right direction is less trivial. Assume that $\langle v \rangle^{LU} \cap Z'$ is empty. Then, there is a negative cycle N in $\min(G_v^{LU}, G_{Z'})$. To prove the proposition, we aim to show the following.

Aim: To show that the negative cycle N of $\min(G_v^{LU}, G_{Z'})$ can be reduced to the form:

$$x \xrightarrow{Z'_{xy}} y \xrightarrow{\langle v \rangle_{yx}^{LU}} x \tag{4.1}$$

Firstly, since both G_v^{LU} and $G_{Z'}$ are canonical, we can assume without loss of generality that no two consecutive edges in N come from the same graph.

Suppose there are two edges $y_1 \to x_1$ and $y_2 \to x_2$ in N with weights coming from G_v^{LU}:

$$y_1 \xrightarrow{\langle v \rangle_{y_1 x_1}^{LU}} x_1 \to \cdots \to y_2 \xrightarrow{\langle v \rangle_{y_2 x_2}^{LU}} x_2 \to \cdots \to y_1 \tag{4.2}$$

Since they are part of a negative cycle, their edge weights should be a finite value and by Part 1 of Lemma 4.3.3, this means:

$$v_{x_1} \leq U_{x_1} \text{ and } v_{x_2} \leq U_{x_2}$$

1. Suppose $v_{y_1} \leq L_{y_1}$ and $v_{y_2} \leq L_{y_2}$. By Part 2 of Lemma 4.3.3, the edge values $y_1 \to x_1$ and $y_2 \to x_2$ are the same as in G_v^M. Consider the edge:

$$y_1 \to x_2 \text{ in } G_v^{LU}$$

Again, from the same lemma, this edge value comes from G_v^M too.

If the value of this edge $y_1 \to x_2$ is smaller than the value of the path $y_1 \to x_1 \to \cdots \to y_2 \to x_2$ in N, then this path can be replaced by the single edge $y_1 \to x_2$ to get a smaller negative cycle in $\min(G_v^{LU}, G_Z)$.

However, if the value of the path $y_1 \to x_1 \to \cdots \to y_2 \to x_2$ is less than the edge value $y_1 \to x_2$, then by Lemma 4.4.2:

$$y_1 \to x_1 \to \ldots \to y_2 \to x_2 \to y_1, \text{ where } x_2 \to y_1 \text{ comes from } G_v^M$$

is a negative cycle. The edge $x_2 \to y_1$ might be infinity in G_v^{LU}. But as G_v^M is canonical, we can replace $y_2 \to x_2 \to y_1 \to x_1$ by $y_2 \to x_1$. From Lemma 4.3.3, this edge is retained in G_v^{LU} and hence we get a smaller negative cycle.

Therefore in this case, we can eliminate the two edges $y_1 \to x_1$ and $y_2 \to x_2$ to get a smaller negative cycle containing either $y_1 \to x_2$ or $y_2 \to x_1$. If N does not contain a variable z such that $v_z > L_z$, this elimination can be repeatedly applied and N can be reduced to a negative cycle of the form $y \to x \to y$ with $v_y \leq L_y$, $v_x \leq U_x$ and the edge weights $y \to x$ coming from G_v^{LU} and $x \to y$ coming from $G_{Z'}$, exactly as required by (4.1).

2. Suppose $v_{y_1} > L_{y_1}$. Consider again the two edges $y_1 \to x_1$ and $y_2 \to x_2$ of (4.2) and now suppose that $v_{y_1} > L_{y_1}$. By Part 3 of Lemma 4.3.3, the edge $y_1 \to x_1$ can be replaced by:

$$y_1 \to x_0 \to x_1 \text{ of } G_v^{LU}$$

If there is another variable in N that is greater than its L bound, then the vertex x_0 would occur twice in the negative cycle. From this negative cycle, we can obtain a smaller negative cycle containing only one occurrence of x_0. Hence, without loss of generality, we can assume that x_0 occurs only once in N. In particular, this gives us that:

$$v_{y_2} \leq L_{y_2}$$

Note that the special variable x_0 has $v_{x_0} \leq L_{x_0}$ as its value is always supposed to be 0 and L_{x_0} is defined to be 0. Now consider the two edges:

$$x_0 \to x_1 \text{ and } y_2 \to x_2$$

This corresponds to Case 1 as $v_{x_0} \leq L_{x_0}$ and $v_{y_2} \leq L_{y_2}$. As we have seen, these two edges can be eliminated to give a smaller negative cycle containing either $x_0 \to x_2$ or $y_2 \to x_1$, with the respective value coming from G_v^{LU}.

If it is the latter edge $y_2 \to x_1$, the smaller negative cycle does not contain y_1 and hence all variables are bounded by L. By Case 1, it can be reduced to a cycle as required by the proposition.

Let us now consider the former edge $x_0 \to x_2$. We have the cycle:

$$y_1 \to x_0 \to x_2 \to \ldots \to y_1$$

All the variables other than y_1 in the path $x_0 \to \ldots \to y_1$ are bounded by their L bound. We can therefore assume that all edges in $x_2 \to \ldots \to y_1$ come from $G_{Z'}$, because if not, we can apply the argument of Case 1 to further reduce the cycle. As $G_{Z'}$ and G_v^{LU} are canonical, this cycle reduces to $y_1 \to x_2 \to y_1$ with $y_1 \to x_2$ coming from G_v^{LU} and $x_2 \to y_1$ coming from Z'. This again conforms to the form of the cycle required by (4.1). □

4.5 Final steps

We will now consider Steps 3 and 4 of Figure 4.1. Proposition 4.4.1 gives a useful characterization of when $\langle v \rangle^{LU} \cap Z'$ is empty. To lift this characterization to $Z \not\subseteq \mathfrak{a}_{\preccurlyeq LU}(Z')$ and Proposition 4.1.2, we need to find the least value of $\langle v \rangle_{yx}^{LU}$ from among the valuations $v \in Z$ and see if this satisfies the condition given in Proposition 4.4.1. Recall that $\langle v \rangle_{yx}^{LU}$ is the weight of the $x - y$ constraint defining the LU-region $\langle v \rangle^{LU}$, in other words, the weight of the edge $y \to x$ in the canonical distance graph G_v^{LU} representing $\langle v \rangle^{LU}$. Before we go ahead looking at the $x - y$ constraints of the LU-regions intersecting a zone, we consider the classic regions first and look at the $x - y$ constraints of the classic regions that intersect a zone. Let us call a line $x - y = c$ an xy-diagonal.

Look at Figure 4.9. We show a zone and the xy-diagonals crossing the zone. The value of $x - y$ decreases as we move towards the *left diagonal boundary* of the zone. The left diagonal boundary is defined by the $y - x$ constraint of Z (c.f. Figures 4.2 and 4.3). The figure (a) on the top has the left boundary closed, that is to say the boundary line is included in the zone. In the bottom one, the boundary line is not included as evident by the strict inequality constraint for $y - x$.

First consider Figure 4.9 (a) with the closed boundary line. The left diagonal boundary is defined by $y - x \leq -1$. In the general case, the weight of the $y - x$ constraint Z_{xy} defining the left boundary will be of the form $(\leq, c)$. Moreover, the least weight of the $x - y$ constraint from among the regions intersecting Z would be $(\leq, -c)$, *the negation of the* $y - x$ *value.*

However, if the boundary line is not included in the zone as in Figure 4.9 (b), we will have Z_{xy} to be $(<, c)$, with a strict inequality. The region with the least value of $x - y$ is the one closest to the left diagonal boundary, as illustrated in the figure. The value of the $x - y$ constraint for this region would be $(<, -c + 1)$, the negation of the $y - x$ value added with one. Due

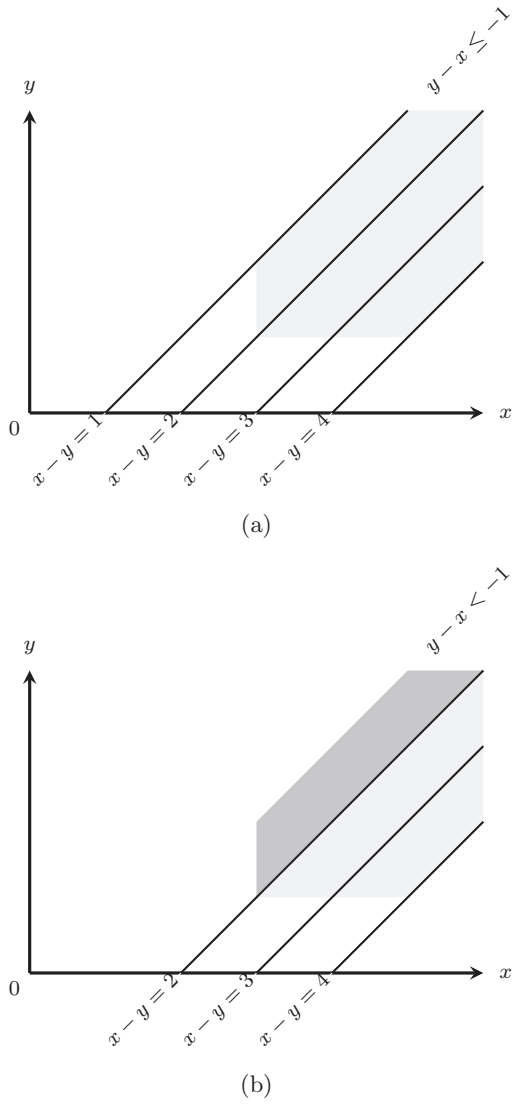


Figure 4.9: Value of the xy-diagonal decreases towards the left.

to this asymmetry, we need to define the following notion to handle weights in a convenient way. We will then use this intuition to get the least value of the xy-diagonal from among the LU-regions.

For a weight $(\prec, c)$ we define $-(\prec, c)$ as $(\prec, -c)$. We now define a *ceiling* function $\lceil \cdot \rceil$ for weights.

Definition 4.5.1 For a real c, let $\lceil c \rceil$ denote the smallest integer that is greater than or equal to c. We define the *ceiling* function $\lceil (\prec, c) \rceil$ for a weight $(\prec, c)$ depending on whether $\prec$ equals $\leq$ or $<$, as follows:

$$\lceil(\leq, c)\rceil = \begin{cases}(\leq, c) & \text{if } c \text{ is an integer}\\ (<, \lceil c\rceil) & \text{otherwise}\end{cases}$$

$$\lceil(<, c)\rceil = \begin{cases}(<, c+1) & \text{if } c \text{ is an integer}\\ (<, \lceil c\rceil) & \text{otherwise}\end{cases}$$

The following lemma is one of the two cores for the proof of the main theorem. It gives the least value of $\langle v\rangle_{yx}^{LU}$ from among the valuations v present in a zone Z.

Proposition 4.5.2 Let Z be a non-empty zone. The least value of $\langle v\rangle_{yx}^{LU}$ for a variable x such that $v_x \leq U_x$ and some other variable y, from among the valuations $v \in Z$ is given by:

$$\begin{cases}(<, \infty) & \text{if } Z_{x0} < (\leq, -U_x)\\ \max\{\lceil -Z_{xy}\rceil, \lceil -Z_{x0}\rceil - (<, L_y)\} & \text{otherwise}\end{cases}$$

Proof
Let G be the canonical distance graph representing the zone Z. We denote the weight of an edge $i \to j$ in G by $(\preccurlyeq_{ij}, c_{ij})$. Recall that this means $Z_{ij} = (\preccurlyeq_{ij}, c_{ij})$.

We are interested in computing the smallest value of the $x-y$ constraint defining an LU-region intersecting Z. Additionally we want to restrict to LU-regions in which all its valuations satisfy $x \leq U_x$, that is, we need to find:

$$\beta := \min\{\langle v\rangle_{yx}^{LU} \mid v \in Z \text{ and } v_x \leq U_x\}$$

We call it β. Clearly, if $v_x > U_x$ for all valuations $v \in Z$, then β is $(<, \infty)$. When $Z_{x0} < (\leq, -U_x)$, it means that all valuations $v \in Z$ satisfy $0 - v_x \preccurlyeq_{x0} c_{x0}$ and $c_{x0} \leq -U_x$. Moreover $\preccurlyeq_{x0}$ is the strict inequality if $c_{x0} = -U_x$. In consequence, all valuations $v \in Z$ satisfy $v_x > U_x$ when $Z_{x0} < (\leq, -U_x)$. Whence $\beta = (<, \infty)$. This corresponds to the first case in the statement of the lemma.

Let now restrict to the case when $Z_{x0} \geq (\leq, -U_x)$. By definition of regions (cf. Definition 4.3) and Lemma 4.3.3, we have for a valuation v:

$$\langle v\rangle_{yx}^{LU} = \begin{cases}\lceil(\leq, v_x - v_y)\rceil & \text{if } v_x \leq U_x \text{ and } v_y \leq L_y\\ (<, -L_y) + \lceil(\leq, v_x)\rceil & \text{if } v_x \leq U_x \text{ and } v_y > L_y\end{cases} \tag{4.3}$$

Let G' be the graph in which the edge $0 \to x$ has weight $\min\{(\leq, U_x), (\preccurlyeq_{0x}, c_{0x})\}$ and the rest of the edges are the same as that of G. This graph G' represents the valuations of Z that have $v_x \leq U_x$: $[\![G']\!] = \{v \in Z \mid v_x \leq U_x\}$. We show that this set is not empty. For this we check that G' does not have negative cycles. Since G does not have negative cycles, every

negative cycle in G' should include the newly modified edge $0 \to x$. Note that the shortest path value from x to 0 does not change due to this modified edge. So the only possible negative cycle in G' is $0 \to x \to 0$. But then we are considering the case when $Z_{x0} \geq (\leq, -U_x)$, and so $Z_{x0} + (\leq, U_x) \geq (\leq, 0)$. Hence this cycle cannot be negative either. In consequence all the cycles in G' are positive and $[\![G']\!]$ is not empty.

To find β, it is sufficient to consider only the valuations in $[\![G']\!]$. As seen from Equation 4.3, among the valuations in $[\![G']\!]$, we need to differentiate between those with $v_y \leq L_y$ and the ones with $v_y > L_y$. We proceed as follows. We first compute $\min\{\langle v \rangle_{yx}^{LU} \mid v \in [\![G']\!] \text{ and } v_y \leq L_y\}$. Call this β_1. Next, we compute $\min\{[v]_{yx} \mid v \in [\![G']\!] \text{ and } v_y > L_y\}$ and set this as β_2. Our required value β would then equal $\min\{\beta_1, \beta_2\}$.

To compute β_1, consider the following distance graph G'_1 which is obtained from G' by just changing the edge $0 \to y$ to $\min\{(\leq, L_y), (\preccurlyeq_{0y}, c_{0y})\}$ and keeping the remaining edges the same as in G'. The set of valuations $[\![G'_1]\!]$ equals $\{v \in [\![G']\!] \mid v_y \leq L_y\}$. If $[\![G'_1]\!] = \emptyset$, we set β_1 to $(<, \infty)$ and proceed to calculate β_2. If not, we see that from Equation 4.3, for every $v \in [\![G'_1]\!]$, $[v]_{yx}$ is given by $\lceil (\leq, v_x - v_y) \rceil$. Let $(\preccurlyeq_1, w_1)$ be the shortest path from x to y in the graph G'_1. Then, we have for all $v \in [\![G'_1]\!]$, $v_y - v_x \preccurlyeq_1 w_1$. If $\preccurlyeq_1$ is $\leq$, then the least value of $[v]_{yx}$ would be $(\leq, -w_1)$ and if $\preccurlyeq_1$ is $<$, one can see that the least value of $[v]_{yx}$ is $(<, -w_1 + 1)$. This shows that $\beta_1 = \lceil (\preccurlyeq_1, -w_1) \rceil$. It now remains to calculate $(\preccurlyeq_1, w_1)$.

Recall that G'_1 has the same edges as in G except possibly different edges $0 \to x$ and $0 \to y$. If the shortest path from x to y has changed in G'_1, then clearly it should be due to one of the above two edges. However note that the edge $0 \to x$ cannot belong to the shortest path from x to y since it would contain a cycle $x \to \ldots 0 \to x \to \ldots y$ that can be removed to give shorter path. Therefore, only the edge $0 \to y$ can potentially yield a shorter path: $x \to \ldots \to 0 \to y$. However, the shortest path from x to 0 in G'_1 cannot change due to the added edges since that would form a cycle with 0 and we know that all cycles in G'_1 are positive. Therefore the shortest path from x to 0 is the direct edge $x \to 0$, and the shortest path from x to y is the minimum of the direct edge $x \to y$ and the path $x \to 0 \to y$. We get: $(\preccurlyeq_1, w_1) = \min\{(\preccurlyeq_{xy}, c_{xy}), (\preccurlyeq_{x0}, c_{x0}) + (\leq, L_y)\}$ which equals $\min\{Z_{xy}, Z_{x0} + (\leq, L_y)\}$. Finally, from the argument in the above two paragraphs, we get:

$$\beta_1 = \begin{cases} (<, \infty) & \text{if } [\![G'_1]\!] = \emptyset \\ \lceil -Z_{xy} \rceil & \text{if } [\![G'_1]\!] \neq \emptyset \text{ and } Z_{xy} \leq Z_{x0} + (\leq, L_y) \\ \lceil -Z_{x0} \rceil + (\leq, -L_y) & \text{if } [\![G'_1]\!] \neq \emptyset \text{ and } Z_{xy} > Z_{x0} + (\leq, L_y) \end{cases} \quad (4.4)$$

We now proceed to compute $\beta_2 = \min\{[v]_{yx} \mid v \in [\![G']\!] \text{ and } v_y > L_y\}$. Let G'_2 be the graph which is obtained from G' by modifying the edge $y \to 0$

to $\min\{Z_{y0}, (<, -L_y)\}$ and keeping the rest of the edges the same as in G'. Clearly $[\![G'_2]\!] = \min\{v \in [\![G']\!] \mid v_y > L_y\}$.

Again, if $[\![G'_2]\!]$ is empty, we set β_2 to $(<, \infty)$. Otherwise, from Equation 4.3, for each valuation $v \in [\![G'_2]\!]$, the value of $[v]_{yx}$ is given by $(<, \lceil v_x \rceil - L_y)$. For the minimum value, we need the least value of v_x from $v \in [\![G'_2]\!]$. Let $(\preccurlyeq_2, w_2)$ be the shortest path from x to 0 in G'_2. Then, since $-v_x \preccurlyeq_2 w_2$, the least value of $\lceil v_x \rceil$ would be $-w_2$ if $\preccurlyeq_2$ is $\leq$ and equal to $\lceil -w_2 \rceil$ if $\preccurlyeq_2$ is $<$ and β_2 would respectively be $(<, -w_2 - L_y)$ or $(<, -w_2 + 1 - L_y)$. It now remains to calculate $(\preccurlyeq_2, w_2)$.

Recall that G'_2 is G with $0 \to x$ and $y \to 0$ modified. The shortest path from x to 0 cannot include the edge $0 \to x$ since it would need to contain a cycle, for the same reasons as in the β_1 case. So we get $(\preccurlyeq_2, w_2) = \min\{Z_{x0}, Z_{xy} + (<, -L_y)\}$. If $Z_{x0} \leq Z_{xy} + (<, -L_y)$, then we take $(\preccurlyeq_2, w_2)$ as Z_{x0}, otherwise we take it to be $Z_{xy} + (<, -L_y)$. So, we get β_2 as the following:

$$\beta_2 = \begin{cases} (<, \infty) & \text{if } [\![G'_2]\!] = \emptyset \\ -Z_{xy} + (<, 1) & \text{if } [\![G'_2]\!] \neq \emptyset \text{ and } Z_{x0} \geq Z_{xy} + (<, -L_y) \\ \lceil -Z_{x0} \rceil + (<, -L_y) & \text{if } [\![G'_2]\!] \neq \emptyset \text{ and } Z_{x0} < Z_{xy} + (<, -L_y) \end{cases} \quad (4.5)$$

However, we would like to write β_2 in terms of the cases used for β_1 in Equation 4.4 so that we can write β, which equals $\min\{\beta_1, \beta_2\}$, conveniently.

Let ψ_1 be the inequation: $Z_{xy} \leq Z_{x0} + (\leq, L_y)$. From Equation 4.4, note that β_1 has been classified according to ψ_1 and $\neg\psi_1$ when $[\![G'_1]\!]$ is not empty. Similarly, let ψ_2 be the inequation: $Z_{x0} \geq Z_{xy} + (<, -L_y)$. From Equation 4.5 we see that β_2 has been classified in terms of ψ_2 and $\neg\psi_2$ when $[\![G'_2]\!]$ is not empty. Notice the subtle difference between ψ_1 and ψ_2 in the weight component involving L_y: in the former the inequality associated with L_y is $\leq$ and in the latter it is $<$. This necessitates a bit more of analysis before we can write β_2 in terms of ψ_1 and $\neg\psi_1$.

Suppose ψ_1 is true. So we have $(\preccurlyeq_{xy}, c_{xy}) \leq (\preccurlyeq_{x0}, c_{x0} + L_y)$. This implies: $c_{xy} \leq c_{x0} + L_y$. Therefore, $c_{x0} \geq c_{xy} - L_y$. When $c_{x0} > c_{xy} - L_y$, ψ_2 is clearly true. For the case when $c_{x0} = c_{xy} - L_y$, note that in ψ_2 the right hand side is always of the form $(<, c_{xy} - L_y)$, irrespective of the inequality in Z_{xy} and so yet again, ψ_2 is true. We have thus shown that ψ_1 implies ψ_2.

Suppose $\neg\psi_1$ is true. We have $(\preccurlyeq_{xy}, c_{xy}) > (\preccurlyeq_{x0}, c_{x0} + L_y)$. If $c_{xy} > c_{x0} + L_y$, then clearly $c_{x0} < c_{xy} - L_y$ implying that $\neg\psi_2$ holds. If $c_{xy} = c_{x0} + L_y$, then we need to have $\preccurlyeq_{xy}$ equal to $\leq$ and $\preccurlyeq_{x0}$ equal to $<$. Although $\neg\psi_2$ does not hold now, we can safely take β_2 to be $\lceil -Z_{x0} \rceil + (<, -L_y)$ as its value is in fact equal to $-Z_{xy} + (<, 1)$ in this case. Summarizing the above

two paragraphs, we can rewrite β_2 as follows:

$$\beta_2 = \begin{cases} (<, \infty) & \text{if } [\![G'_2]\!] = \emptyset \\ -Z_{xy} + (<, 1) & \text{if } [\![G'_2]\!] \neq \emptyset \text{ and } Z_{xy} \leq Z_{x0} + (\leq, L_y) \\ \lceil -Z_{x0} \rceil + (<, -L_y) & \text{if } [\![G'_2]\!] \neq \emptyset \text{ and } Z_{xy} > Z_{x0} + (\leq, L_y) \end{cases} \quad (4.6)$$

We are now in a position to determine β as $\min\{\beta_1, \beta_2\}$. Recall that we are in the case where $Z_{x0} \leq (\leq, -U_x)$ and we have established that $[\![G']\!]$ is non-empty. Now since $[\![G']\!] = [\![G'_1]\!] \cup [\![G'_2]\!]$ by construction, both of them cannot be simultaneously empty. Hence from Equations 4.4 and 4.6, we get β, the $\min\{\beta_1, \beta_2\}$ as:

$$\beta_= \begin{cases} \lceil -Z_{xy} \rceil & \text{if } Z_{xy} \leq Z_{x0} + (\leq, L_y) \\ \lceil -Z_{x0} \rceil + (<, -L_y) & \text{if } Z_{xy} > Z_{x0} + (\leq, L_y) \end{cases} \quad (4.7)$$

There remains one last reasoning. To prove the lemma, we need to show that $\beta = \max\{\lceil -Z_{xy} \rceil, \lceil -Z_{x0} \rceil + (<, -L_y)\}$. For this it is enough to show the following two implications:

$$Z_{xy} \leq Z_{x0} + (\leq, L_y) \Rightarrow \lceil -Z_{xy} \rceil \geq \lceil -Z_{x0} \rceil + (<, -L_y)$$
$$Z_{xy} > Z_{x0} + (\leq, L_y) \Rightarrow \lceil -Z_{xy} \rceil \leq \lceil -Z_{x0} \rceil + (<, -L_y)$$

We prove only the first implication. The second follows in a similar fashion. Let us consider the notation $(\prec_{xy}, c_{xy})$ and $(\prec_{x0}, c_{x0})$ for Z_{xy} and Z_{x0} respectively. So we have:

$$\begin{aligned} & (\prec_{xy}, c_{xy}) \leq (\prec_{x0}, c_{x0}) + (\leq, L_y) \\ \Rightarrow \; & (\prec_{xy}, c_{xy}) \leq (\prec_{x0}, c_{x0} + L_y) \end{aligned}$$

If the constant $c_{xy} < c_{x0} + L_y$, then $-c_{xy} > -c_{x0} - L_y$ and we clearly get that $\lceil -Z_{xy} \rceil \geq \lceil -Z_{x0} \rceil + (<, -L_y)$. If the constant $c_{xy} = c_{x0} + L_y$ and if $\prec_{x0}$ is $\leq$, then the required inequation is trivially true; if $\prec_{x0}$ is $<$, it implies that $\prec_{xy}$ is $<$ too and clearly $\lceil (<, -c_{xy}) \rceil$ equals $\lceil (<, -c_{x0}) \rceil + (<, -L_y)$. □

We have now established Step 2 and 3 of the schema shown in Figure 4.1. We have a simple method that tells us when an LU-region $\langle v \rangle^{LU}$ does not intersect a zone Z' (Proposition 4.4.1). We have also characterized the potential valuation v from Z that could satisfy the non-intersection condition with Z' (Proposition 4.5.2). This gives the necessary tools to solve the final Step 4 of Figure 4.1. The following theorem presents the efficient inclusion test $Z \not\subseteq \mathfrak{a}_{\preccurlyeq LU}(Z')$.

Theorem 4.5.3 *Let Z, Z' be non-empty zones. Then, $Z \not\subseteq \mathfrak{a}_{\preccurlyeq LU}(Z')$ iff there exist two variables x, y such that:*

$$Z_{x0} \geq (\leq, -U_x) \text{ and } Z'_{xy} < Z_{xy} \text{ and } Z'_{xy} + (<, -Ly) < Z_{x0}$$

Proof
From Proposition 4.1.2, we know that $Z \not\subseteq \mathfrak{a}_{\preccurlyeq LU}(Z')$ iff there exists a valuation $v \in Z$ such that $\langle v \rangle^{LU}$ does not intersect Z'.

From Proposition 4.4.1, we know that $\langle v \rangle^{LU} \cap Z'$ is empty iff there exists a variable x such that $v_x \leq U_x$ and a variable y such that:

$$Z'_{xy} + \langle v \rangle^{LU}_{yx} < (\leq, 0) \tag{4.8}$$

This is possible for variables x, y iff the least value of $\langle v \rangle^{LU}_{yx}$ from among the valuations in Z satisfies the inequation (4.8) with Z'_{xy}.

This is where we use Proposition 4.5.2. According to this proposition, for (4.8) to be true for some valuation $v \in Z$, we would need $Z_{x0} \geq (\leq, -U_x)$ and:

$$Z'_{xy} + \lceil -Z_{xy} \rceil < (\leq, 0) \text{ and } Z'_{xy} + \lceil -Z_{x0} \rceil - (<, L_y) < (\leq, 0) \tag{4.9}$$

Consider the first inequality: $Z'_{xy} + \lceil -Z_{xy} \rceil < (\leq, 0)$. Let Z_{xy} be $(\prec_{xy}, c_{xy})$. If $\prec_{xy}$ is the weak inequality $\leq$, then $\lceil -Z_{xy} \rceil$ is $(\leq, -c_{xy})$ and hence the condition becomes: $Z'_{xy} + (\leq, -c_{xy}) < (\leq, 0)$. This is equivalent to saying $Z'_{xy} < (\leq, c_{xy})$, that is, $Z'_{xy} < Z_{xy}$. Now, if $\prec_{xy}$ is the strict inequality $<$, then $\lceil -Z_{xy} \rceil$ becomes $(<, -c_{xy} + 1)$ and hence the condition becomes: $Z'_{xy} + (<, -c_{xy} + 1) < (\leq, 0)$. This is equivalent to saying $Z'_{xy} < (<, c_{xy})$. In both cases, the first inequality of Equation (4.9) becomes $Z'_{xy} < Z_{xy}$.

By a similar reasoning, the second inequality of Equation (4.9) can be seen to correspond to $Z'_{xy} + (<, -L_y) < Z_{x0}$. This proves the theorem. □

Clearly, the test involves a comparison of corresponding edges in the distance graphs G_Z and $G_{Z'}$ and takes a worst case of $\mathcal{O}(|X|^2)$ number of steps. Notice that in fact the test requires only two tests for every pair of clocks.

4.6 Efficient inclusion $Z \subseteq \mathit{Closure}_M(Z')$

As a continuation to the discussion in Section 3.4, from Theorem 4.5.3, we get the inclusion test $Z \subseteq \mathit{Closure}_M(Z')$ as a bonus. By definition, $\mathit{Closure}_M(Z')$ is the union of $[v']^M$ for all $v' \in Z'$. When $L = U = M$, the LU-region $\langle v \rangle^{LU}$ becomes the classic region $[v]^M$. To know if $Z \not\subseteq \mathit{Closure}_M(Z')$, we need to know if there exists a valuation $v \in Z$ such that v does not belong to $[v']^M$ for every valuation $v' \in Z'$. It can be easily seen that this is equivalent to saying no valuation $v' \in Z'$ belongs to the region

$[v]^M$, that is $[v]^M \cap Z'$ is empty. This gives us a characterization exactly similar to Proposition 4.1.2. We thus get the inclusion test for $Closure_M$ by just substituting $L = U = M$ in the above theorem.

Corollary 4.6.1 Let Z, Z' be non-empty zones. Then, $Z \not\subseteq Closure_M(Z')$ iff there exist variables x, y such that:

$$Z_{x0} \geq (\leq, -M_x) \text{ and } Z'_{xy} < Z_{xy} \text{ and } Z'_{xy} + (<, -M_y) < Z_{x0}$$

4.7 Concluding remarks

In this chapter, we have given an efficient inclusion test $Z \not\subseteq \mathfrak{a}_{\preccurlyeq LU}(Z')$ in Theorem 4.5.3 that has the same complexity as $Z \not\subseteq Z'$. Not only the complexities are the same, the test is a slight modification of the edge by edge comparison used for $Z \not\subseteq Z'$. To know if $Z \not\subseteq Z'$ we need to check if there are x, y such that $Z'_{xy} < Z_{xy}$. In the $\mathfrak{a}_{\preccurlyeq LU}$ inclusion, in addition to this there are two tests: $Z_{x0} \geq (\leq, -U_x)$ and $Z'_{xy} + (<, -L_y) < Z_{x0}$. This shows that the test $Z \not\subseteq \mathfrak{a}_{\preccurlyeq LU}(Z')$ can be implemented as efficiently as $Z \not\subseteq Z'$.

We now summarize the important steps leading to our result. A pictorial representation has already been given in Figure 4.1. We will now list in more detail.

1. We first modified the question of inclusion $Z \not\subseteq \mathfrak{a}_{\preccurlyeq LU}(Z')$ to a question of intersection in Proposition 4.1.2. The proposition states that $Z \not\subseteq \mathfrak{a}_{\preccurlyeq LU}(Z')$ iff there exists $v \in Z$ such that $\langle v \rangle^{LU} \cap Z'$ is empty.

2. We then proceeded to study the intersection $\langle v \rangle^{LU} \cap Z'$. Our first step was to define a convenient representation to study this intersection. To this regard, we defined distance graphs in Section 4.2 and characterized when a distance graph has an empty solution set in Proposition 4.2.2. Given distance graphs G_1, G_2, we denoted by $\min(G_1, G_2)$ the graph representing the intersection $[\![G_1]\!] \cap [\![G_2]\!]$. From Proposition 4.2.2, the intersection $[\![G_1]\!] \cap [\![G_2]\!]$ is empty iff $\min(G_1, G_2)$ has a negative cycle.

3. For a zone Z', we know the canonical distance graph representation, say $G_{Z'}$. In Definition 4.3.1, we gave the canonical distance graph G_v^{LU} representation of an LU-region $\langle v \rangle^{LU}$ so that we could use the result of the previous step.

4. Having the distance graphs $G_{Z'}$ and G_v^{LU}, we analyzed $\min(G_{Z'}, G_v^{LU})$ for negative cycles in Section 4.4 and came up with Proposition 4.4.1 that says that there is a negative cycle iff there are two variables x, y with $v_x \leq U_x$ such that $Z'_{xy} + \langle v \rangle^{LU}_{yx} < (\leq, 0)$. Thus, this intersection reduces to looking at pairs of clocks.

5. The final step involved transferring Proposition 4.4.1 and Proposition 4.1.2 to the final goal $Z \not\subseteq \mathfrak{a}_{\preccurlyeq LU}(Z')$. In Proposition 4.5.2, we give the least value of $\langle v \rangle_{yx}^{LU}$ from among the valuations $v \in Z$ with $v_x \leq U_x$. If $Z \not\subseteq \mathfrak{a}_{\preccurlyeq LU}(Z')$ then this least value should satisfy the condition $Z'_{xy} + \langle v \rangle_{yx}^{LU} < (\leq, 0)$. We use this value and end up with the efficient inclusion test in Theorem 4.5.3.

6. As a corollary to $\mathfrak{a}_{\preccurlyeq LU}$ inclusion, we also get the $Closure_M$ inclusion test by substituting $L = U = M$ (Corollary 4.6.1).

From Theorem 4.5.3, one can see that the larger the L and U bounds, the higher the chance that $Z \not\subseteq \mathfrak{a}_{\preccurlyeq LU}(Z')$. Therefore, if one gets as small LU-bounds as possible, one could expect more inclusions to happen during the reachability algorithm. Of course, if a zone Z is included into $\mathfrak{a}_{\preccurlyeq LU}(Z')$ of an already visited zone Z, the algorithm does not further explore Z. One therefore hopes that the algorithm terminates faster when the LU-bounds are smaller. This is the topic of the next chapter where we see how to obtain tight LU-bounds.

Chapter 5

Tightening the bounds

Previous chapters have shown that the $\mathfrak{a}_{\preccurlyeq LU}$ abstraction is the coarsest possible abstraction when using only LU-bounds. We have also seen how to use the $\mathfrak{a}_{\preccurlyeq LU}$ abstraction efficiently in implementations. The optimality shows that one cannot do better than the $\mathfrak{a}_{\preccurlyeq LU}$ abstraction if we have only the LU-bounds. The question now is if we can do something about the LU-bounds themselves. As we noted in the concluding remarks of the previous chapter, the test for $\mathfrak{a}_{\preccurlyeq LU}$ (Theorem 4.5.3) shows that the bigger the LU-bounds, the greater is the possibility for non-inclusion. Therefore, the smaller the LU-bounds, the better are the chances for inclusion, and hence one would hope to get a smaller reachability tree. The goal of this chapter is to get LU-bounds as tight as possible so that reachability tree we obtain is a still sound approximation of the behaviour.

In Section 2.6, we recalled the state-of-the-art method to compute bounds by performing a static analysis on the automaton. By this method, a bound function is assigned to every state q of the automaton. Paths in the automaton starting from q are analyzed. All guards that occur before a corresponding reset are considered and the maximum constant among these guards is assigned at state q. Automaton $\mathcal{A}_{stat}$ shown in Figure 2.14 gives very good gains by using the static analysis approach.

However, one could do better. Consider the automaton shown in Figure 5.1.

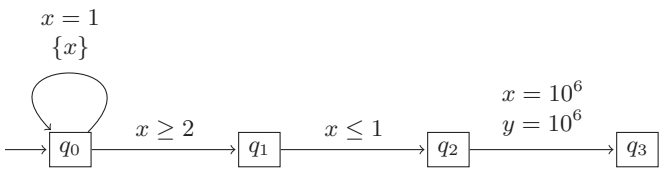


Figure 5.1: Timed automaton $\mathcal{A}_{sem}$.

Automaton $\mathcal{A}_{sem}$ does not have resets of clocks x and y on the transition from q_0 to q_1 as in the automaton of Figure 2.14. By the static analysis ap-

proach, one would get $L_y(q_0) = U_y(q_0) = 10^6$. Same for clock x. This gives once again an order of 10^6 nodes in the zone graph. However, a closer look reveals that the state q_2 is not at all reachable from q_0. We need to know the semantics. But we do not know this unless we run the reachability algorithm. In the following section, we will show how to calculate better bounds on-the-fly during the exploration of the zone graph. The bounds we calculate make use of the extra information gained during the exploration procedure. We have implemented this on-the-fly constant propagation algorithm along with the coarser $\mathfrak{a}_{\preccurlyeq LU}$ abstraction discussed in the previous chapter. We validate the gains of our proposed improvements by experimental results on some standard verification case studies (c.f.Table 5.1).

Organization of the chapter

There are two sections in this chapter. Section 5.1 describes a new algorithm for reachability that calculates bounds on-the-fly during exploration. In Section 5.2, we report on some experiments conducted with the new algorithm and the $\mathfrak{a}_{\preccurlyeq LU}$ abstraction. We compare the results the new algorithm with the state-of-the-art algorithm that uses the static analysis approach to calculate bounds, and uses the convex abstraction $Extra^+_{LU}$.

5.1 Constant propagation

We can improve on the idea of static analysis (c.f. Section 2.6) that computes a bound function for each state q. We will compute these bounding functions on-the-fly and they will depend also on a zone and not just a state. We wish to associate bound functions LU for every node (q, Z) in the zone graph $ZG(\mathcal{A})$. We look at paths starting from (q, Z) in $ZG(\mathcal{A})$ and search for a relevant constant that appears before a corresponding reset.

$$(q, Z) \to \underbrace{(q_1, Z_1) \to \ldots \to (q_n, Z_n)}_{\text{no resets of } x} \xrightarrow{x \geq c} (q', Z')$$

An obvious gain is that we will never consider constraints coming from unreachable transitions. But the question is how do we know the constants before actually running the reachability algorithm. This is the main challenge in the algorithm that we propose below.

Our modified algorithm is given in Algorithm 5.1. It computes a tree whose nodes are triples (q, Z, LU) where (q, Z) is a node of $ZG(\mathcal{A})$ and LU are bound functions. Each node (q, Z, LU) has as many child nodes (q_s, Z_s, L_sU_s) as there are successors (q_s, Z_s) of (q, Z) in $ZG(\mathcal{A})$. Notice that this includes successors with an empty zone Z_s, which are however not further unfolded. These nodes must be included for correctness of our constant propagation procedure. By default bound functions map each clock

Algorithm 5.1: Reachability algorithm with on-the-fly bound computation and $\mathfrak{a}_{\preccurlyeq LU}$ abstraction.

```
1  function main():
2        Add (q_0, Z_0, L_0U_0) to the stack
3        while (stack ≠ ∅) do
4             Remove (q, Z, LU) from the stack
5             explore(q, Z, LU)
6             resolve()
7        return "empty"
8
9  function explore(q, Z, LU):
10       if (q is accepting)
11            exit "not empty"
12       if (∃ (q, Z', L'U') nontentative s.t. Z ⊆ a_{≼L'U'}(Z'))
13            mark (q, Z, LU) tentative wrt (q, Z', L'U')
14            LU := L'U'
15            propagate(parent(q, Z, LU))
16       else
17            propagate(q, Z, LU)
18            for each (q_s, Z_s, L_sU_s) s.t. (q, Z, LU) ⇒ (q_s, Z_s, L_sU_s) do
19                 if (Z_s ≠ ∅)
20                      explore(q_s, Z_s, L_sU_s)
21
22 function resolve():
23       for each (q, Z, LU) tentative w.r.t. (q, Z', L'U') do
24            if (Z ⊈ a_{≼L'U'}(Z'))
25                 mark (q, Z, LU) nontentative
26                 LU := −∞;
27                 propagate(parent(q, Z, LU))
28                 Add (q, Z, LU) to stack
29
30 function propagate(q, Z, LU):
31       LU := max_{(q,Z,LU) ⇒^{g;R} (q',Z',L'U')} maxedge(g, R, L'U')
32       if (LU has changed)
33            for each (q_t, Z_t, L_tU_t) tentative wrt (q, Z, LU) do
34                 L_tU_t := LU;
35                 propagate(parent(q_t, Z_t, L_tU_t))
36            if ((q, Z, LU) ≠ (q_0, Z_0, L_0U_0))
37                 propagate(parent(q, Z, LU))
38
39 function maxedge(g, R, LU):
40       let L_R = λx. if x ∈ R then −∞ else L_x
41       let U_R = λx. if x ∈ R then −∞ else U_x
42       let L_g = λx. if x > c or x ≥ c in g then c else −∞
43       let U_g = λx. if x < c or x ≤ c in g then c else −∞
44       L = λx. max(L_R(x), L_g(x))
45       U = λx. max(U_R(x), U_g(x))
46       return (LU)
```

to $-\infty$. They are later updated as explained below. Each node is further marked either *tentative* or *nontentative*. The leaf nodes (q, Z, LU) of the tree are either deadlock nodes (either there is no transition out of state q or Z is empty), or *tentative* nodes. All the other nodes are marked *nontentative*.

Our algorithm starts from the root node (q_0, Z_0, L_0U_0), consisting of the initial state, initial zone, and the function mapping each clock to $-\infty$. It repeatedly alternates an *exploration* and a *resolution* phase.

Exploration phase

Before exploring a node $n = (q, Z, LU)$ the function `explore` checks if q is accepting and Z is not empty; if it is so then $\mathcal{A}$ has an accepting run. Otherwise the algorithm checks if there exists a *nontentative* node $n' = (q', Z', L'U')$ in the current tree such that $q = q'$ and $Z \subseteq \mathfrak{a}_{\preccurlyeq L'U'}(Z')$. If yes, n becomes a *tentative* node and its exploration is temporarily stopped as each state reachable from n is also reachable from n', with respect to the current values $L'U'$ of n'. If none of these holds, the successors of the node are explored. The exploration terminates since $\mathfrak{a}_{\preccurlyeq LU}$ has a finite range.

When the exploration algorithm gets to a new node, it propagates the bounds from this node to all its predecessors. The goal of these propagations is to maintain the following invariant. For every node $n = (q, Z, LU)$:

1. if n is *nontentative*, then α is the maximum of the L_sU_s from all successor nodes (q_s, Z_s, L_sU_s) of n (taking into account guards and resets as made precise in the function `maxedge`);

2. if n is *tentative* with respect to $(q', Z', L'U')$, then LU is equal to $L'U'$.

The result of propagation is analogous to the inequalities seen in the static guard analysis [BBFL03], however now applied to the zone graph, on-the-fly. Hence, the bounds associated to each node (q, Z, LU) never exceed those that are computed by the static guard analysis.

A delicate point about this procedure is handling of tentative nodes. When a node n is marked *tentative*, we have $LU = L'U'$. Indeed, exploring n would have resulted in bounds LU not bigger than $L'U'$ and hence n is covered by $(q', Z', L'U')$. However the value of $L'U'$ may be updated when the tree is further explored. Thus each time we update the bounds function of a node n', it is not only propagated upward in the tree but also to the nodes that are tentative with respect to n'. When bounds get updated, n may not be covered by n' anymore. This is taken care of in the resolution phase.

Each exploration phase terminates as in the `explore` procedure the bound functions in each node never decrease and are bounded. From the invariants above, we get that in every node, LU is a solution to the equations in (2.3) applied on $ZG(\mathcal{A})$.

It could seem that the algorithm will be forced to do a high number of propagations of bounds. The experiments reported in Section 5.2 show that the present very simple approach to bound propagation is good enough. Since we propagate the bounds as soon as they are modified, most of the time, the value of LU does not change in line 31 of function `propagate`. In general, bounds are only propagated on very short distances in the tree, mostly along one single edge. For this reason we do not concentrate on optimizing the function `propagate`. In the implementation we use the presented function augmented with a minor "optimization" that avoids calculating maximum over all successors in line 31 when it is not needed.

Resolution phase

Finally, as the bounds may have changed since n has been marked tentative, the function `resolve` checks for the consistency of *tentative* nodes. If $Z \subseteq \mathfrak{a}_{\preccurlyeq L'U'}(Z')$ is not true anymore, n needs to be explored. Hence it is viewed as a new node: the bounds are set to $-\infty$ and n is pushed on the *stack* for further consideration in the function `main`. Setting α to $-\infty$ is safe as α will be computed and propagated when n is explored. We perform also a small optimization and propagate this bound upward, thereby making some bounds decrease.

The resolution phase may provide new nodes to be explored. The algorithm terminates when this is not the case, that is when all tentative nodes remain tentative. We can then conclude that no accepting state is reachable. Note that the overall algorithm should terminate, that is, at some point of time, all the tentative nodes should remain tentative. As we have seen, the bounds in a node (q, Z) are not bigger than the bounds obtained for q by static analysis. So at some point of time, every tentative node should be covered by a non-tentative node, as $\mathfrak{a}_{\preccurlyeq LU}$ is a finite abstraction.

Theorem 5.1.1 *An accepting state is reachable in $ZG(\mathcal{A})$ iff the algorithm reaches a node with an accepting state and a non-empty zone.*

The right to left direction is straightforward, so we concentrate on the opposite direction. The left to right implication of the theorem follows from the next lemma.

Lemma 5.1.2 For every (q, Z) reachable in $ZG(\mathcal{A})$, there exists a non tentative node (q, Z_1, L_1U_1) in the tree constructed by the LU-algorithm, such that $Z \subseteq \mathfrak{a}_{\preccurlyeq L_1U_1}(Z_1)$.

Proof
The hypothesis is vacuously true for (q_0, Z_0). Assume that the hypothesis is true for a node $(q, Z) \in ZG(\mathcal{A})$. We prove that the lemma is true for every successor of (q, Z).

From hypothesis, there exists a node (q, Z_1, L_1, U_1) in the tree constructed by Algorithm 5.1 such that $Z \subseteq \mathfrak{a}_{\preccurlyeq_{L_1U_1}}(Z_1)$. Let $t = (q, g, r, q')$ be a transition of $\mathcal{A}$ and let $(q, Z) \Rightarrow^t (q', Z') \in ZG(\mathcal{A})$. There are two cases.

(q, Z_1) **is not tentative** The scenario of this case is depicted in the figure below:

$$\begin{array}{ccc} ZG(\mathcal{A}) & & \text{Algorithm 5.1} \\ & \overset{Z \subseteq \mathfrak{a}_{\preccurlyeq_{L_1U_1}}(Z')}{\dashrightarrow} & \\ (q, Z) & & (q, Z_1, L_1U_1) \\ \Downarrow t & & \Downarrow t \\ & \overset{\text{Is } Z' \subseteq \mathfrak{a}_{\preccurlyeq_{L_1'U_1'}}(Z_1')?}{\dashrightarrow} & \\ (q', Z') & & (q', Z_1', L_1'U_1') \end{array}$$

We know that $Z \subseteq \mathfrak{a}_{\preccurlyeq_{L_1U_1}}(Z_1)$. We want to prove that $Z' \subseteq \mathfrak{a}_{\preccurlyeq_{L_1'U_1'}}(Z_1')$. For this, we need to show that for every valuation $v' \in Z'$ there is a valuation $v_1' \in Z_1'$ that is simulating v' with respect to the bounds $L_1'U_1'$, that is:

$$v' \preccurlyeq_{L_1'U_1'} v_1'$$

Recall the definition of the transition relation $\Rightarrow^t$ given in Definition 2.3.1. According to the definition:

$$Z' = \{v' \mid \exists v \in Z, \exists \delta \in \mathbb{R}_{\geq 0} : v \rightarrow^t \rightarrow^\delta v'\}$$

Let v_a' be a valuation in Z' such that it is obtained from Z by the action transition $\rightarrow^t$:

$$\text{there exists } v \in Z \text{ such that } v \rightarrow^t v_a'$$

By hypothesis, we have $Z \subseteq \mathfrak{a}_{\preccurlyeq_{L_1U_1}}(Z_1)$. Hence for the valuation v that belongs to Z, there exists $v_1 \in Z_1$ such that:

$$v \preccurlyeq_{L_1U_1} v_1$$

Since the transition $\rightarrow^t$ is enabled from v, it is enabled from v_1 too. The successor of v after $\rightarrow^t$ is v_a'. Let $v_1 \rightarrow^t v_1'$. Since $\preccurlyeq_{L_1U_1}$ is a simulation relation, we have:

$$v_a' \preccurlyeq_{L_1U_1} v_1'$$

We now want to show that the above simulation is true with respect to the bounds $L'_1U'_1$. From the invariants of Algorithm 5.1, we know that for all x not reset in the transition t, we have $L'_1(x) \leq L_1(x)$ and $U'_1(x) \leq U_1(x)$. For all x reset in the transition, we have $v'_a(x) = v'_1(x) = 0$. Therefore, from Definition 2.5.1 of the $\preccurlyeq_{LU}$ simulation relation, we get that:

$$\begin{array}{rcl} v'_a & \preccurlyeq_{L'_1U'_1} & v'_1 \\ v'_a + \delta & \preccurlyeq_{L'_1U'_1} & v'_1 + \delta \end{array}$$

for all $\delta \in \mathbb{R}_{\geq 0}$. Recall that v'_a is a valuation obtained from some $v \in Z$ by an action transition. By the definition of $\Rightarrow^t$, every valuation in Z' is of the form $v'_a + \delta$. Thus, we get that $Z' \subseteq \mathfrak{a}_{\preccurlyeq_{L'_1U'_1}}(Z'_1)$.

(q, Z_1) **is tentative** If it is a tentative node, we know that there exists a non-tentative node (q, Z_2, L_2U_2) in the tree constructed by the LU-algorithm such that $Z_1 \subseteq \mathfrak{a}_{\preccurlyeq_{L_2U_2}}(Z_2)$. The rest of the argument is the same as in the previous case with (q, Z_2, L_2U_2) instead of (q, Z_1, L_1U_1).

□

5.2 Experiments

We have implemented the $\mathfrak{a}_{\preccurlyeq_{LU}}$ inclusion test (Theorem 4.5.3) and the constant propagation Algorithm 5.1 on a prototype tool and have tested it on classical benchmarks. The results are presented in Table 5.1, where the effect of the $\mathfrak{a}_{\preccurlyeq_{LU}}$ abstraction and on-the-fly bounds are compared separately. Detailed explanations follow.

Let us consider columns 4-5 showing the effect of the $\mathfrak{a}_{\preccurlyeq_{LU}}$ abstraction. The FDDI protocol shows a noticeable gain due to the $\mathfrak{a}_{\preccurlyeq_{LU}}$ abstraction. The situation observed on the FDDI protocol is explained in Figure 5.2 on the left. For the zone Z in the figure, by definition $Extra^+_{LU}(Z') = Z'$, and in consequence $Z \not\subseteq Extra^+_{LU}(Z')$. However, $Z \subseteq \mathfrak{a}_{\preccurlyeq_{LU}}(Z')$. This leads to quicker inclusions and hence fewer nodes. Note that in the FDDI protocol with n processes, the DBMs are rather big square matrices of order $3n + 2$. But still, our inclusion test based on $\mathfrak{a}_{\preccurlyeq_{LU}}$ is competitive in the running time. In the other benchmarks that we have tried, the $\mathfrak{a}_{\preccurlyeq_{LU}}$ abstraction does not seem to give a significant gain, but nevertheless the inclusion test does not take much extra time either. More importantly, this inclusion test and the structure of the algorithm has enabled the application of the constant propagation algorithm which gives us substantial gains.

We now consider the columns 6-7 that tabulate the experiments performed with the constant propagation algorithm. The improvement comes from the on-the-fly computation of the LU-bounds as demonstrated by the examples $\mathcal{A}_2$ (Figure 5.2), Fischer and CSMA/CD that correspond to three

Model	$Extra^+_{LU}$, sa		$\mathfrak{a}_{\preccurlyeq LU}$, sa		$\mathfrak{a}_{\preccurlyeq LU}$, otf		UPPAAL	
	nodes	s.	nodes	s.	nodes	s.	nodes	s.
$\mathcal{A}_1$	10003	0.09	10002	0.11	2	0.00	10003	0.07
$\mathcal{A}_2$	2003	0.32	2002	0.73	6	0.00	2003	0.01
$\mathcal{A}_3$	10004	0.39	10003	0.81	3	0.00	10004	0.32
CSMA/CD7	5923	0.38	5923	0.40	5832	0.48	-	T.O.
CSMA/CD8	19017	1.48	19017	1.60	18701	2.00	-	T.O.
CSMA/CD9	60783	5.91	60783	6.48	59816	10.65	-	T.O.
FDDI10	525	0.04	421	0.04	421	0.04	12049	2.43
FDDI20	2045	0.62	1641	0.64	1641	0.60	-	T.O.
FDDI30	4565	3.16	3661	3.46	3661	3.10	-	T.O.
Fischer7	18353	0.56	18353	0.61	11372	0.56	18374	0.35
Fischer8	85409	2.80	85409	3.14	39412	2.30	85438	1.53
Fischer9	397989	14.04	397989	16.38	133503	9.17	398685	8.95
Fischer10	-	T.O.	-	T.O.	-	T.O.	1827009	53.44

Table 5.1: Effect of $\mathfrak{a}_{\preccurlyeq LU}$ and on-the-fly bounds listed separately. Columns 2-3 show results of the state-of-the-art algorithm with $Extra^+_{LU}$ abstraction, and bounds by static analysis (Section 2.6). Columns 4-5 change the abstraction to $\mathfrak{a}_{\preccurlyeq LU}$ and keep the bounds from static analysis. Columns 6-7 consider both the $\mathfrak{a}_{\preccurlyeq LU}$ abstraction and on-the-fly bounds (Algorithm 5.1). Columns 8-9 consider the tool UPPAAL 4.1.3 with options (-n4 - C -01). Experimental results: number of visited nodes and running time with a timeout (T.O.) of 60 seconds. Experiments done on a MacBook with 2.4GHz Intel Core Duo processor and 2GB of memory running MacOS X 10.6.7.

different situations. In the $\mathcal{A}_2$ example, the transition that yields the big bounds $L_y = U_y = 10^4$ on y in q_0 is not reachable from any (q_0, Z), hence we just get the lower bound $L_y = 20$ on y in (q_0, Z), and a subsequent gain in performance.

The automaton $\mathcal{A}_1$ in Figure 5.2 illustrates the gain on the CSMA/CD protocol. The transition from q_0 to q_1 is disabled as it must synchronize on letter $a!$. The static analysis algorithm ignores this fact, hence it associates bound $L_y = 10^4$ to y in q_0. Since our algorithm computes the bounds on-the-fly, y is associated only the bound $U_y = 10$ in every node (q_0, Z). We observe that algorithms using static analysis visit 10003 nodes on $ZG(\mathcal{A}_1)$ whereas using on-the-fly bounds needs to visit only 2 nodes. The same situation occurs in the CSMA/CD example. However despite the improvement in the number of nodes the cost of computing the bounds impacts the running time negatively.

The gains that we observe in the analysis of the Fischer's protocol are explained by the automaton $\mathcal{A}_3$ in Figure 5.2. $\mathcal{A}_3$ has a bounded integer variable n that is initialized to 0. Hence, the transitions from q_0 to q_2, and from q_1 to q_2, that check if n is equal to 10 are disabled. This is ignored by the static analysis algorithm that associates the bound $L_y = 10^4$ to clock y in q_0. Our algorithm however associates only the bound $U_y = 10$ to y in every node (q_0, Z). We observe that static analysis requires visits to 10004 nodes whereas on-the-fly constant propagation algorithm only visits 3 nodes.

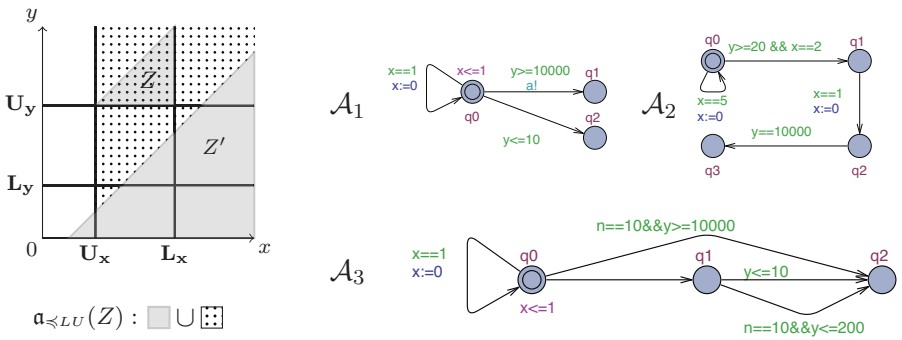


Figure 5.2: Examples explaining gains obtained with the algorithm.

A similar situation occurs in the Fischer's protocol.

We have also included a comparison of our algorithm with the tool UPPAAL itself. In columns 8-9 we present the results of UPPAAL 4.1.3 with the options (-n4 -C -o1). Since we have not considered symmetry reduction [HBL+04] in our tool, we have not used it in UPPAAL either.

The comparison to UPPAAL is not meaningful for the CSMA/CD and the FDDI protocols. Indeed, UPPAAL runs out of time even if we significantly increase the time allowed; switching to breadth-first search has not helped either. We suspect that this is due to the order in which UPPAAL takes the transitions in the automaton. For this reason in columns 1 and 2, we provide results from our own implementation of UPPAAL's algorithm that takes transitions in the same order as the implementation of our algorithm. We include the last row to underline that our implementation is not as mature as UPPAAL. We strongly think that UPPAAL could benefit from methods presented here. Although RED [Wan04] also uses approximations, it is even more difficult to draw a meaningful comparison with it, since it uses symbolic state representation unlike UPPAAL or our tool. Since this thesis is about approximation methods, and not tool comparison, we leave more extensive comparisons as further work.

5.3 Concluding remarks

In this chapter, we have seen a new algorithm to obtain bounds on-the-fly during the exploration of the zone graph (Algorithm 5.1). We have proved the correctness of our approach in Theorem 5.1.1. The static analysis approach assigns bounds to every state q of the automaton; in our approach, we assign bounds to every (q, Z) of the zone graph. The bounds we assign to each (q, Z) are never bigger than the ones obtained by static analysis for the state q. The immediate gain is that we do not consider guards on transitions from unreachable states.

We have discussed in Section 5.2 the gains observed by our algorithm using $\mathfrak{a}_{\preccurlyeq LU}$ abstraction and on-the-fly bounds computation. We see that most of the observed gains come from on-the-fly bounds. These on-the-fly bounds help to prune the search to a considerable extent. We believe that over bigger models, the gains would be much more conspicuous. These gains also show that for future, one could try to use more information from the reachability tree than just LU-bounds to cut the search sooner.

On the other hand, it remains to understand better the effect of $\mathfrak{a}_{\preccurlyeq LU}$. In the examples that we have considered, $\mathfrak{a}_{\preccurlyeq LU}$ has a gain in one of them. In the future, we would like to find examples where $\mathfrak{a}_{\preccurlyeq LU}$ wins over its convex counterpart by a bigger margin. This would in particular help understand the extra power of the non-convex $\mathfrak{a}_{\preccurlyeq LU}$ abstraction in the reachability analysis.

Part II

Liveness

Chapter 6

Preliminaries

In this part, we consider infinite behaviours of timed automata. Since timed automata model reactive systems that continuously interact with the environment, it is interesting to consider questions related to their infinite executions. In particular, the liveness question asks if some property occurs repeatedly throughout the execution of the automaton. A standard way to model liveness is by using a Büchi condition.

While considering infinite executions, one has to eliminate the so-called *Zeno* runs. A run is said to be Zeno if an infinite number of events happen in a finite time interval. During verification of a liveness property, one asks if there exists an infinite run that violates the property. If this run turns out to be a Zeno run, then it is not a realistic counter-example to the property. Therefore, the aim during verification of a liveness property is to detect if there exists a *non-Zeno* run that violates the property. The first goal of this part of the thesis is to study this problem: given a timed automaton, does there exist a non-Zeno run satisfying the Büchi accepting condition. We call this the *Büchi non-emptiness problem* for timed automata.

As we saw in the previous part, state reachability has been handled by making use of regions, zones and simulation graphs (Sections 2.2, 2.3, 2.4). This has been possible since every finite run of the automaton corresponds to a finite run of the simulation graph and vice versa. It has been shown that the convex abstractions of Figure 2.12 are sound and complete even for repeated state reachability, that is, every infinite run of the automaton corresponds to an infinite run of the simulation graph and vice versa [TYB05, Li09]. To detect non-Zeno runs, the automaton can be transformed to what is called a *strongly non-Zeno* automaton [TYB05]. This transformation adds an extra clock to take care of non-Zenoness. Our problem then reduces to checking emptiness of an untimed Büchi automaton. This problem has been extensively studied. The best known algorithms for solving this problem rely on a variant of Tarjan's algorithm for detecting strongly connected components, known as the Couvreur's algorithm [CDLP05, Cou99, GS09].

On a parallel note, while implementing timed automata, it is required to check for the presence of pathological *Zeno* runs. Note that this is not the negation of the non-Zenoness problem. For this case, a sufficient-only condition on the syntax of the automaton that ensures the absence of Zeno runs has been proposed [GB07].

Organization of the chapter

This chapter brings together the literature related to the problem of detecting infinite runs of the automaton and classifying them as non-Zeno or Zeno. We start with the notion of timed Büchi automata in Section 6.1 and define the problems that we are interested in. In Section 6.2, we will see that regions are sound and complete for repeated state reachability. In the next Section 6.3, we will see this is true for zones and abstractions defined using a time-abstract simulation. Subsequently, we recall the strongly non-Zeno construction in Section 6.4. This is followed by Section 6.5 which details the techniques known for the detection of Zeno-runs. We end this chapter with an outline (Section 6.6) that summarizes our results presented the following chapters.

6.1 Büchi non-emptiness problem

Let X be a set of clocks. We will now consider infinite runs of a timed automaton. Let $\mathcal{A} = (Q, q_0, X, T, Acc)$ be a timed automaton as in Definition 2.1.1.

An *infinite run* of $\mathcal{A}$ is an infinite sequence of configurations connected by transitions, starting from the initial state q_0 and the initial valuation $v_0 = \mathbf{0}$:

$$(q_0, v_0) \xrightarrow{\delta_0, t_0} (q_1, v_1) \xrightarrow{\delta_1, t_1} \cdots$$

While considering infinite runs, we need to define a different notion of acceptance. We consider the Büchi accepting condition.

Definition 6.1.1 (Büchi condition, Zeno runs and non-Zeno runs) A run σ *satisfies the Büchi condition* if it visits *accepting configurations* infinitely often, that is configurations with a state from Acc. The *duration* of the run is the accumulated delay: $\sum_{i \geq 0} \delta_i$. An infinite run σ is *Zeno* if it has a finite duration. Otherwise, it is *non-Zeno*.

The problem we are interested is termed the *Büchi non-emptiness problem*.

Definition 6.1.2 (Büchi non-emptiness) The *Büchi non-emptiness problem* is to decide if $\mathcal{A}$ has a non-Zeno run satisfying the Büchi condition.

The Büchi non-emptiness problem is known to be PSPACE-complete thanks to [AD94]. As in the case of reachability, we consider diagonal free timed automata, where clock constraints like $x - y \leq 1$ are disallowed. Since we are interested in the Büchi non-emptiness problem, we can consider automata without an input alphabet. Also, we consider automata without invariants since they can be simulated by guards.

Remark 6.1.3 (Timed Büchi automata) In this part of the thesis, we refer to timed automata as *timed Büchi automata (TBA)* whenever we talk about the Büchi non-emptiness problem.

6.2 Regions

We have seen in Chapter 2 that the analysis of timed automata requires an abstraction of its uncountable space of configurations into a finite graph that captures the behaviour of the automaton with respect to a certain property. For reachability, we have seen different abstractions in the previous part of the thesis. We will now see the different abstractions that are sound and complete with respect to Büchi emptiness.

Recall the notion of regions and region graph $RG(\mathcal{A})$ as defined in Section 2.2 for timed automata. We use the same definitions for a timed Büchi automaton. By definition of the region graph, for every run of the automaton, there is a run of the region graph. This is true even for infinite runs. In Lemma 2.2.7, we showed that the transitions in the region graph are pre-stable. This allowed us to conclude that every path in the region graph can be instantiated to a run of the timed automaton. Lemma 2.2.8 proves it for finite paths. However, the same argument holds for infinite paths of the region graph too. We begin by defining the notions of abstraction and instantiation for infinite runs.

Consider two sequences

$$(q_0, v_0) \xrightarrow{\delta_0, t_0} (q_1, v_1) \xrightarrow{\delta_1, t_1} \cdots \tag{6.1}$$

$$(q_0, r_0) \xrightarrow{t_0} (q_1, r_1) \xrightarrow{t_1} \cdots \tag{6.2}$$

where the first is a run in $\mathcal{A}$, and the second is a path in $RG(\mathcal{A})$. We say that the first is an *instantiation* of the second if $v_i \in r_i$ for all $i \geq 0$. Equivalently, we say that the second is an *abstraction* of the first. The following lemma is a direct consequence of the pre-stability property of Lemma 2.2.7.

Lemma 6.2.1 Given a TBA $\mathcal{A}$, Every path in $RG(\mathcal{A})$ is an abstraction of a run of $\mathcal{A}$, and conversely, every run of $\mathcal{A}$ is an instantiation of a path in $RG(\mathcal{A})$.

The above lemma shows that the region graph is sound and complete for repeated state reachability. It also allows us to relate the existence of an accepting run of $\mathcal{A}$ to the existence of paths with special properties in $RG(\mathcal{A})$. We say that a path as in (6.2) *satisfies the Büchi condition* if it has infinitely many occurrences of states from Acc.

Definition 6.2.2 (Progressive path [AD94]) For an automaton $\mathcal{A}$ let $M : X \mapsto \mathbb{N} \cup \{-\infty\}$ be the bound function assigning to each clock the maximum constant appearing in a guard of $\mathcal{A}$ (c.f. Definition 2.2.5). Let $RG(\mathcal{A})$ be the region graph of $\mathcal{A}$ defined using the bound function M. A path of $RG(\mathcal{A})$ as in (6.2) is called *progressive* if for every clock $x \in X$:

- either x is almost always above M_x: there is n with $r_i \vDash x > M_x$ for all $i > n$;
- or x is reset infinitely often and strictly positive infinitely often: for every n there are $i, j > n$ such that $r_i \vDash (x = 0)$ and $r_j \vDash (x > 0)$.

It has been shown that a progressive path of the region graph instantiates to a non-Zeno run of the automaton and vice-versa.

Theorem 6.2.3 ([AD94]) *A TBA $\mathcal{A}$ has a non-Zeno run satisfying the Büchi conditions iff $RG(\mathcal{A})$ has a progressive path satisfying the Büchi condition.*

Moreover, the progress criterion above can be encoded by adding an extra Büchi accepting condition [AD94, TYB05]. Although this gives a procedure to solve the Büchi non-emptiness problem, the number of regions obtained using a bound function M is $\mathcal{O}\big(|X|! \cdot 2^{|X|} \cdot \prod_{x \in X}(2M_x + 2)\big)$ [AD94]. This makes this solution impractical.

6.3 Zones and abstractions

As in the case of reachability, we resort to zones for solving the Büchi non-emptiness problem. In Section 2.3, we defined a symbolic semantics for timed automata called its zone graph $ZG(\mathcal{A})$ (Definition 2.3.4). The zone graph could potentially be infinite, as illustrated by the example of Figure 2.8. Therefore, we need to abstract zones further using abstraction operators. An abstraction operator defines an abstract symbolic semantics called the simulation graph $SG^{\mathfrak{a}}(\mathcal{A})$ (Definition 2.4.2). We have seen different abstractions that are sound and complete for reachability in Section 2.5.

We have classified abstraction operators to fall into broadly two classes: M-abstractions and LU-abstractions, which use maximal bounds and LU-bounds respectively as parameters. A more detailed explanation about M-abstractions and LU-abstractions can be found in Section 2.4. It has been

shown in [TYB05] that the M-abstractions are sound and complete for repeated state reachability. Few years later, it has been shown in [Li09] that even the LU-abstractions are sound and complete for repeated state reachability. To handle non-Zenoness from simulation graphs, Tripakis *et al* have proposed a construction adding an extra clock to the automaton [TYB05]. This non-Zenoness construction is the topic of the next section. In this section, we will recall the proof of [Li09] that every abstraction that is defined using a time-abstract simulation is sound and complete for repeated state reachability. All the standard abstractions (Figure 2.12) are defined using a simulation relation and hence turn out to be sound and complete even for infinite runs.

We recall below the definitions of time-abstract simulation and abstractions based on simulations (Definitions 2.4.4 and 2.4.5). Let us assume a given automaton $\mathcal{A}$. The transition system $\mathcal{S}_\mathcal{A}$ describing the semantics of $\mathcal{A}$ has been defined in Definition 2.1.3.

►**Definition 2.4.4. (Time-abstract simulation)** A *(state based) time-abstract simulation* between two states of transition system $\mathcal{S}_\mathcal{A}$ is a relation $(q, v) \preceq_{t.a.} (q', v')$ such that:

- $q = q'$,
- if $(q, v) \to^{\delta} (q, v + \delta) \to^{t} (q_1, v_1)$, then there exists a $\delta' \in \mathbb{R}_{\geq 0}$ such that $(q', v') \to^{\delta'} (q', v' + \delta') \to^{t} (q'_1, v'_1)$ satisfying $(q_1, v_1) \preceq_{t.a.} (q'_1, v'_1)$ for the same transition t.

For two valuations v, v', we say that $v \preceq_{t.a.} v'$ if for every state q of the automaton, we have $(q, v) \preceq_{t.a.} (q', v')$.

►**Definition 2.4.5. (Abstraction based on simulation)** Given a zone Z, we define $\mathfrak{a}_{\preceq_{t.a.}}(Z) = \{v \mid \exists v' \in Z.\ v \preceq_{t.a.} v'\}$.

In the following two lemmas, we show that a cycle in the simulation graph can be instantiated to an infinite run of the automaton.

Lemma 6.3.1 Let $\mathcal{A}$ be a TBA and let $\mathfrak{b}$ be an abstraction operator defined using a time-abstract simulation $\preceq_\mathfrak{b}$. Let:

$$(q_1, W_1) \Rightarrow_\mathfrak{b}^{t_1} (q_2, W_2) \Rightarrow_\mathfrak{b}^{t_2} \dots (q_n, W_n)$$

be a path in the simulation graph $SG^\mathfrak{b}(\mathcal{A})$. Then, for every valuation $v_n \in W_n$ there is a valuation $v'_1 \in W_1$ and $\delta_1, \delta_2, \dots \in \mathbb{R}_{\geq 0}$ such that:

$$(q_1, v'_1) \xrightarrow{\delta_1, t_1} (q_2, v'_2) \xrightarrow{\delta_2, t_2} \dots (q_n, v'_n)$$

is a run of $\mathcal{A}$ with $v_n \preceq_\mathfrak{b} v'_n$ and $v'_i \in W_i$ for all $i \in \{1, \dots, n\}$

Proof
We prove the lemma by induction.

Base case. For the base case, consider a single transition $(q_1, W_1) \Rightarrow_{\mathfrak{b}}^{t_1} (q_2, W_2)$ and let $v_2 \in W_2$ be a valuation. By definition of transitions in the simulation graph, there exist $v_1' \in W_1$ and $\delta_1' \in \mathbb{R}_{\geq 0}$ such that:

$$(q_1, v_1') \rightarrow^{t_1} (q_2, v) \rightarrow^{\delta_1'} (q_2, v_2') \text{ and } v_2 \preceq_{\mathfrak{b}} v_2'$$

As $\preceq_{\mathfrak{b}}$ is a *time-abstract* simulation relation, the intermediary valuation v can simulate v_2, that is: $v_2 \preceq_{\mathfrak{b}} v$. This gives us a run $(q_1, v_1) \xrightarrow{\delta_1, t_1} (q_2, v)$ with $\delta_1 = 0$. This is of the required form, thus proving the base case.

Induction case. Let us now consider the induction case. Let $v_n \in W_n$ be a valuation. For some $i \in \{2, \ldots, n-1\}$, let $v_i'' \in W_i$ and let there exist the following path:

$$(q_i, v_i'') \rightarrow^{t_i} \rightarrow^{\delta_i''} \ldots (q_n, v_n'') \tag{6.3}$$

such that $v_n \preceq_{\mathfrak{b}} v_n''$. Consider the transition $(q_{i-1}, W_{i-1}) \Rightarrow_{\mathfrak{b}}^{t_{i-1}} (q_i, W_i)$ and the valuation $v_i'' \in W_i$. Applying the argument of the base case to this transition, we get a valuation $v_{i-1}' \in W_{i-1}$ and $\delta_{i-1}' \in \mathbb{R}_{\geq 0}$ such that:

$$(q_{i-1}, v_{i-1}') \rightarrow^{t_{i-1}} \rightarrow^{\delta_{i-1}'} (q_i, v_i')$$

and $v_i'' \preceq_{\mathfrak{b}} v_i'$. By definition of time-abstract simulation, we will get that there is a path corresponding to (6.3) from (q_i, v_i'):

$$(q_{i-1}, v_{i-1}') \rightarrow^{t_{i-1}} \rightarrow^{\delta_{i-1}} (q_i, v_i') \rightarrow^{t_i} \rightarrow^{\delta_i'} \ldots (q_n, v_n')$$

and $v_n'' \preceq_{\mathfrak{b}} v_n'$. By transitivity of $\preceq_{\mathfrak{b}}$, we get that $v_n \preceq_{\mathfrak{b}} v_n'$. It can be seen that the above path can be converted to the run of the required form in a straightforward manner. □

Lemma 6.3.2 Let $\mathcal{A}$ be a TBA and let $\mathfrak{b}$ be an abstraction operator defined using a time-abstract simulation $\preceq_{\mathfrak{b}}$. Let:

$$(q, W) \Rightarrow_{\mathfrak{b}}^{t_1} \ldots \Rightarrow_{\mathfrak{b}}^{t_n} (q, W)$$

be a path in the simulation graph $SG^{\mathfrak{b}}(\mathcal{A})$. Then, there exists a valuation $v \in W$ such that starting from (q, v) there is a run of $\mathcal{A}$ visiting state q infinitely often.

Proof
We first introduce some notations. We write $(q, W) \Rightarrow_{\mathfrak{b}}^{1 \ldots n} (q, W)$ to denote the path: $(q, W) \Rightarrow_{\mathfrak{b}}^{t_1} \ldots \Rightarrow_{\mathfrak{b}}^{t_n} (q, W)$. Similarly we write $(q, v) \rightarrow^{1 \ldots n} (q, v')$ to denote that there exists a run $(q, v) \xrightarrow{\delta_1, t_1} \ldots \xrightarrow{\delta_n, t_n} (q, v')$.

Consider the given simulation graph path: $(q, W) \Rightarrow_{\mathfrak{b}}^{1\ldots n} (q, W)$. For some index $i \in \mathbb{N}$ let $v_i \in W$. By Lemma 6.3.1, there exists a valuation v'_{i-1} such that:

$$(q, v'_{i-1}) \rightarrow^{1\ldots n} (q, v'_i)$$

is a run of the automaton and $v_i \preceq_{\mathfrak{b}} v'_i$. Using the fact that $\mathfrak{b}$ is based on a simulation relation and applying the above inductively, we can get valuations $v'_1, \ldots, v'_{i-1}$ such that:

$$(q_1, v'_1) \rightarrow^{1\ldots n} (q_2, v'_2) \rightarrow^{1\ldots n} \ldots (q, v'_{i-1}) \rightarrow^{1\ldots n} (q, v'_i) \tag{6.4}$$

is a run of the automaton and $v_i \preceq_{\mathfrak{b}} v'_i$. Note that this is possible for an arbitrary $i \in \mathbb{N}$.

Let M be the maximal bounds function for $\mathcal{A}$ and $\mathcal{R}_M$ denotes the set of regions obtained using this bound function. In particular, if we take $i > |\mathcal{R}_M|$, then in the run given by (6.4), there are two valuations v'_k and v'_l such that $v'_k \sim_M v'_l$ and there is a run of the automaton from (q, v'_k) to (q, v'_l). Therefore, there is a cycle in the region graph with the node (q, r) where r is the region containing v'_k and v'_l. By Lemma 2.2.8, we get an infinite run of the automaton that visits state q infinitely often. □

This brings us to the main result of the section that simulation graphs constructed using abstraction operators based on some time-abstract simulation are sound and complete for repeated state reachability.

Theorem 6.3.3 ([Li09]) *Let $\mathcal{A}$ be a TBA and $\mathfrak{b}$ an abstraction operator defined using a time-abstract simulation. Then, $\mathcal{A}$ has a run satisfying the Büchi condition iff $SG^{\mathfrak{b}}(\mathcal{A})$ has a path satisfying the Büchi condition.*

Proof
If $\mathcal{A}$ has a run satisfying the Büchi condition, then by completeness of abstraction $\mathfrak{b}$, there exists a path of $SG^{\mathfrak{b}}(\mathcal{A})$ that satisfies the Büchi condition.

For the converse, suppose there is a path of $SG^{\mathfrak{b}}(\mathcal{A})$ that satisfies the Büchi condition. This means that there is a cycle in $SG^{\mathfrak{b}}(\mathcal{A})$ with a node (q, W) where $q \in Acc$. Moreover, (q, W) is reachable from the initial node (q_0, W_0). By Lemma 6.3.2, there exists $v \in W$ and a run of $\mathcal{A}$ starting from (q, v) satisfying the Büchi condition. As (q, W) is reachable in $SG^{\mathfrak{b}}(\mathcal{A})$, there exists a path from the initial valuation $(q_0, \mathbf{0})$ to (q, v') such that $v \preceq_{\mathfrak{b}} v'$. But, by definition of simulation relation, there exists a path from (q, v') that satisfies the Büchi condition. □

Although the above generic theorem shows that all the standard abstractions defined in literature (c.f. Figure 2.12), are sound and complete for repeated state reachability, only convex abstractions are used in implementations. However, in this part of the thesis we do not focus on the usage

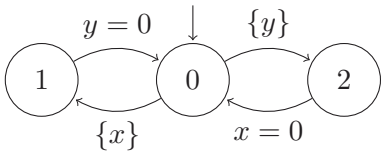


Figure 6.1: Automaton $\mathcal{A}_{zeno}$ with only Zeno runs

$$(0, 0 = x = y) \rightarrow (1, 0 = x = y) \rightarrow (0, 0 = x = y) \rightarrow (2, 0 = y = x) \prec (0, 0 = x = y) \cdots$$

Figure 6.2: A part of the region graph for the automaton $\mathcal{A}_{zeno}$

$$(0, 0 = x = y) \rightarrow (1, 0 = x \leq y) \rightarrow (0, 0 = x = y) \rightarrow (2, 0 = y \leq x) \rightarrow (0, 0 = x = y) \cdots$$

Figure 6.3: A part of the simulation graph for the automaton. $\mathcal{A}_{zeno}$

of non-convex abstractions in the implementations of liveness checking algorithms. Instead, the goal of this part of the thesis is to look more closely at another subtle point: how does one efficiently determine *non-Zenoness* from these abstract simulation graphs.

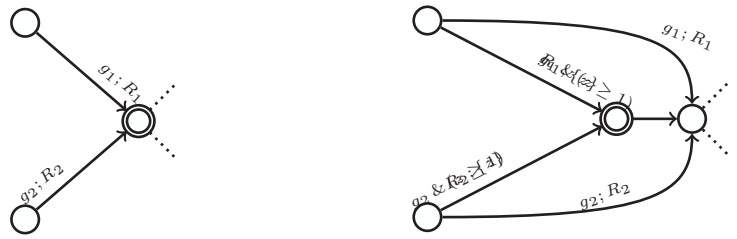
Remark 6.3.4 (Notation) Since all the known abstraction operators we deal with are based on time-abstract simulation, in the subsequent part of the thesis, whenever we refer to a generic simulation graph $SG^{\mathfrak{a}}(\mathcal{A})$, it is assumed that $\mathfrak{a}$ is an abstraction defined using a time-abstract simulation.

6.4 Existence of non-Zeno runs

Theorem 6.3.3 tells that every path in the simulation graph can be instantiated to a path of the automaton. However, it does not guarantee that a path we find in a simulation graph has an instantiation that is non-Zeno. This cannot be decided from a simulation graph $SG^{\mathfrak{a}}(\mathcal{A})$ by using the progress criterion of Definition 6.2.2 as we show now.

Consider for instance the automaton $\mathcal{A}_{zeno}$ in Figure 6.1 which has only Zeno runs as both x and y must remain equal to 0 on every run. Figure 6.2 shows a part of $RG(\mathcal{A}_{zeno})$. The infinite path starting from node $(0, 0 = x = y)$ is not progressive as none of the clocks can have a positive value. Moreover, it can be seen that every node where a clock has a positive value is a deadlock node.

Figure 6.3 depicts the corresponding part of $SG^{\mathfrak{a}}(\mathcal{A}_{zeno})$ with the abstraction operator $\mathfrak{a}$ taken to be $Extra_M$, the most "precise" abstraction (Figure 2.12). This path satisfies the progress criterion as both x and y are



A **SNZ(A)**

Figure 6.4: Strongly non-Zeno construction (z is a new clock).

reset and may have positive values infinitely often, despite all its instantiations being Zeno. The progress criterion fails due to the loss of pre-stability in $SG^{\mathfrak{a}}(\mathcal{A}_{zeno})$: valuations with either $x > 0$ or $y > 0$ which do not have any successor. Note that all other abstractions contain $Extra_M$ and hence suffer from the same problem.

Strongly non-Zeno construction

A common solution to deal with Zeno runs is to transform an automaton into one where all runs satisfying the Büchi condition are guaranteed to be non-Zeno. The automaton obtained as a result of this transformation is called the *strongly non-Zeno* automaton. Consider an automaton $\mathcal{A}$. The main idea behind the transformation of $\mathcal{A}$ into a strongly non-Zeno automaton $SNZ(\mathcal{A})$ is to ensure that on every accepting run, time elapses for 1 time unit infinitely often. Hence, it is sufficient to check for the existence of an accepting run as it is non-Zeno for granted.

The transformation depicted in Figure 6.4 adds one clock z and duplicates accepting states. One copy is no longer accepting whereas the other is accepting, but it can be reached only when $z \geq 1$. Moreover, when an accepting state is reached z is reset to 0.

Definition 6.4.1 (Strongly non-Zeno automaton [TYB05]) Given a timed automaton $\mathcal{A} = (Q, q_0, X, T, Acc)$, the *strongly non-Zeno* automaton $SNZ(\mathcal{A})$ is given by $(Q', q_0, X \cup \{z\}, T', Acc)$ where $z \notin X$ and Q' includes the states Q along with a state q'_{acc} for every state $q_{acc} \in Acc$. This state q'_{acc} does not belong to Acc. For every transition $t = (q, g, R, q_{acc})$ to an accepting state q_{acc}, we add the transition (q, g, R, q'_{acc}) and modify the guard and reset of t to give $(q, g \wedge z \geq 1, R \cup \{z\}, q_{acc})$. All transitions $(q_{acc}, -, -, -)$ are replaced by $(q'_{acc}, -, -, -)$ and an extra transition $(q_{acc}, true, \emptyset, q'_{acc})$ is added.

As a result, every accepting run in $SNZ(\mathcal{A})$ has a corresponding run in $\mathcal{A}$ where every occurrence of q_{acc} is replaced by an occurrence of either itself

or its non-accepting copy q'_{acc}. Since two occurrences of the accepting copy q_{acc} have to be separated by at least one time unit, every accepting run in $SNZ(\mathcal{A})$ is necessarily non-Zeno. This leads us to the following theorem.

Theorem 6.4.2 *[TYB05] Automaton $\mathcal{A}$ has an accepting non-Zeno run iff $SNZ(\mathcal{A})$ has an accepting run.*

Therefore, to solve the Büchi non-emptiness problem for a TBA $\mathcal{A}$, the standard solution is to construct the strongly non-Zeno TBA $SNZ(\mathcal{A})$ and analyze the simulation graph of this strongly non-Zeno automaton.

6.5 Existence of Zeno runs

Another interesting problem relevant for the implementation of timed automata is the question of existence of Zeno runs. Given an automaton, does there exist a Zeno run? A bulk of the literature directs to work of Bowman and Gomez. In [BG06], they provide a syntactic criterion on the automaton that is sufficient to ensure the absence of Zeno runs. This procedure is inspired by the strongly non-Zeno construction of Tripakis *et al.* However it is a sufficient-only criterion, that is, there are examples where the criterion is not satisfied but still Zeno runs are absent. We recall this criterion below, and show an example that does not satisfy this criterion and yet is free from Zeno runs.

A *loop* lp of an automaton is a sequence of states and edges: $q_1 \xrightarrow{g_1,R_1} q_2 \xrightarrow{g_2,R_2} \ldots \xrightarrow{g_n,R_n} q_n$ such that $q_1 = q_n$. We denote by $Guards(lp)$ the set $\{g_1, \ldots, g_n\}$ of guards appearing in the transitions of lp. We define $Resets(lp)$ to be the set of clocks reset in some transition of the loop. The following definition is inspired by the construction of the previous section.

Definition 6.5.1 A loop lp of an automaton is *strongly non-Zeno* if there is a clock $x \in Resets(lp)$ that is bounded from below in some guard $g \in Guards(lp)$: that is $g \vDash x \geq 1$.

It can then be proved that if all the loops in an automaton are strongly non-Zeno, then the automaton has no Zeno runs.

Lemma 6.5.2 [Gom06, BG06] If all loops of an automaton $\mathcal{A}$ are strongly non-Zeno, then $\mathcal{A}$ does not have Zeno runs.

However it is sufficient-only due to a very simple reason. Consider the automaton shown in Figure 6.5. The automaton has no strongly non-Zeno loop, but yet there are no Zeno runs: the loop cannot be completed at all. We see that there is a need to consider information about the executions of the automaton, and not just the automaton.

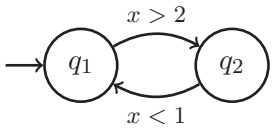


Figure 6.5: Example showing an automaton with a loop that is not strongly non-Zeno, yet there are no Zeno runs

6.6 Outline

In this part of the thesis, we consider infinite runs of timed automata. We have recalled that even to detect infinite runs, one can use simulation graphs thanks to Theorem 6.3.3 (note that all the abstractions we consider are based on time-abstract simulation). The goal of this part of the thesis is to study the constructions used for detecting non-Zeno runs and Zeno runs. Before we elaborate on our contributions, we start with an important remark about the conventions adopted for the subsequent part of the thesis.

Remark 6.6.1 In this part of the thesis, we consider only convex abstractions. Hence we choose to call simulation graphs as *abstract zone graphs* in this rest of the thesis. Additionally we do not consider the optimization of bounds using static analysis (c.f. Section 2.6). Instead we assume global bound functions common to all states of the automaton. The results of this part can be easily extended to static analysis bounds. As regards the new techniques developed for non-convex abstractions and on-the-fly bounds propagation in Chapters 3, 4 and 5, we postpone their adaptability to liveness for future work.

We provide better methods to detect existence of non-Zeno runs. These new methods give us an improved algorithm for the Büchi non-emptiness problem. For the case of Zeno runs, we give the first complete solution to detect existence of Zeno runs in an automaton. More details follow.

Non-Zenoness problem

The current procedure for detecting non-Zenoness modifies the automaton by adding an auxiliary clock. Although the modification is simple, we give an example where this leads to an abstract zone graph that is exponentially bigger than the abstract zone graph of the original automaton. This is the starting point of our work and leads us to investigating solutions that avoid this blowup. We propose a solution that does not modify the automaton and instead works with the abstract zone graph directly (Theorem 7.2.8). We notice that this construction gives a polynomial complexity for M-abstractions (Theorem 7.3.5). However, we notice that even this new construction could give an exponential blowup for the LU-abstractions. Further investigation

reveals that the problem of deciding if an automaton has a non-Zeno run given its LU-abstract zone graph as input is NP-complete (Theorem 7.4.2). As a final observation related to the non-Zenoness problem, we see that weakening the LU-abstraction slightly gives polynomial complexity.

Efficient emptiness check for timed Büchi automata

An important module in the emptiness check for TBA is the procedure to check non-Zenoness. The analysis in the previous chapter shows that the standard construction is costly. We provide an on-the-fly algorithm that starts by exploring the abstract zone graph of the original automaton and uses our new non-Zenoness construction only when it is necessary. We see that in most cases, non-Zenoness can be verified directly without any extra construction. We justify with some benchmarks.

Zenoness problem

As a final goal, we consider the problem of detecting zeno runs. Currently there is only a partial solution to this problem. This solution looks at the automaton and gives a sufficient condition for the absence of zeno runs. We propose a complete solution to this problem by considering the abstract zone graph of the automaton (Theorem 9.1.6). The solution has an extra linear cost for the M-abstractions. However for the LU-abstractions, the solution might cost an extra exponential. Yet again, it turns out that this problem is NP-complete for the LU-abstractions (Theorem 9.2.3). We study the reason for this blowup and propose a weakening of the LU-abstractions that avoids this blowup.

Chapter 7

Non-Zenoness problem

We are interested in the Büchi non-emptiness problem for timed automata which asks if a given timed automaton has a non-Zeno run satisfying the Büchi accepting condition. To solve this problem, one could analyze either the region graph or an abstract zone graph (see Remark 6.6.1 for terminology). To be able to use these graphs, one should first know if an infinite path in these graphs instantiates to an infinite run of the automaton and vice-versa. We have seen that this property is true for region graph and abstract zone graphs using the standard abstractions. However, one cannot guarantee if the instantiation yields a non-Zeno run of the automaton.

In the case of the region graph, this problem can be solved by checking a simple progressiveness property on the path obtained [AD94]. This property can conveniently be encoded as a Büchi condition. A property of this kind works for region graphs because of the pre-stability condition (Lemma 2.2.7): if a transition can be taken from one valuation of a region, the transition can be taken from every valuation of the region. Such a property holds no more for zones and abstracted zones and hence the progressiveness property is no longer sound for zones and abstract zones. To take care of non-Zenoness, the standard solution is to modify the automaton instead. An auxiliary clock is added to ensure that there is at least a unit time elapse between two visits to an accepting state occurring infinitely often. This trick known as the strongly non-Zeno construction is currently the state-of-the-art for non-Zenoness checking [TYB05]. We have recalled this construction in Section 6.4. Given an automaton, the algorithm first converts it to its strongly non-Zeno counterpart and then analyzes the abstract zone graph of this strongly non-Zeno automaton.

In this chapter, we closely examine this strongly non-Zeno construction. We observe that this seemingly simple transformation could potentially lead to an exponential blowup to the size of the abstract zone graph. Subsequently, we propose a new construction and analyze its complexity with respect to different abstractions.

Organization of the chapter

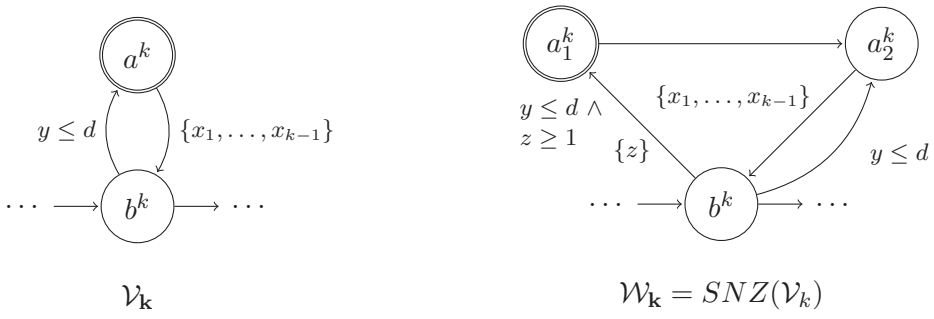
We elucidate below the goals of the chapter and the corresponding sections that discuss these goals.

Goal 1. The first goal of this chapter is to show that the standard trick of adding a clock, although very simple, might lead to an automaton with an abstract zone graph that is exponentially bigger than the abstract zone graph of the original automaton. Additionally, this holds for any standard abstraction operator that is used. The problem comes from the fact that this additional clock allows to remember distances between clocks and this could proliferate to give a large number of zones. We explain this in more detail in Section 7.1.

Goal 2. The second goal of this chapter is to propose a construction for non-Zenoness that avoids the blowup. We propose a construction that modifies the abstract zone graph instead of modifying the automaton. The construction that we propose is called the guessing zone graph construction. We prove that this construction is correct for detecting non-Zeno runs (Theorem 7.2.8). We detail this construction in Section 7.2.

Goal 3. The next goal would be to analyze the complexity of this proposed construction. As it deals with a modification of the abstract zone graph, one could expect the complexity to depend on the specific abstraction used. We will see that this construction applied directly avoids the blowup only for the simplest M-abstraction (Theorem 7.3.5). So as a next step, we modify the construction slightly to get what is called the reduced guessing zone graph. We characterize the abstractions that would maintain a low polynomial complexity using this construction (Theorem 7.3.11). In particular, both the M-abstractions satisfy this criterion. This would be the subject of Section 7.3.

Goal 4. The coarser LU-abstractions do not satisfy the criterion for polynomial complexity using the reduced guessing zone graph. The final goal of this chapter would be to investigate this marked fact that both the strongly non-Zeno and the guessing zone graph constructions blowup the LU-abstract zone graphs exponentially. A further analysis shows that this blowup for the LU-abstractions is possibly unavoidable. In Section 7.4 we show that given the automaton and the abstract zone graph with respect to LU-abstractions, deciding if the automaton has a non-Zeno run is NP-complete (Theorem 7.4.2). We propose a weakening of the LU-abstractions that can avoid NP-completeness.



$\mathcal{V}_\mathbf{k}$ $\mathcal{W}_\mathbf{k} = SNZ(\mathcal{V}_k)$

Figure 7.1: The gadgets $\mathcal{V}_k$ (left) and $\mathcal{W}_k = SNZ(\mathcal{V}_k)$ (right).

We direct the reader to Section 7.5 on concluding remarks for a detailed summary of this chapter, including references to important definitions and results.

7.1 Adding a clock leads to exponential blowup

We discuss why the strongly non-Zeno construction of Section 6.4, although simple, may add an exponential factor in the algorithm. A slightly different construction is mentioned in [AM04]. Of course one can also have other modifications, and it is impossible to treat all the imaginable constructions at once. Our objective here is to show that the constructions proposed in the literature produce a phenomenon causing proliferation of zones that can sometimes be exponential in the number of clocks. The discussion below will focus on the construction from [TYB05], but the one from [AM04] suffers from the same problem.

The strongly non-Zeno construction of Section 6.4 is illustrated by an example in Figure 7.1. The problem comes from the fact that the constraint $z \geq 1$ may be a source of rapid multiplication of the number of zones in the zone graph of $SNZ(\mathcal{A})$. We will now show an example of an automaton where this rapid multiplication indeed occurs. Before we proceed to the example, we wish to give the following remark about the bounds and abstractions considered.

Remark 7.1.1 For ease of presentation, we will assume that the bound function M associates the same constant to every clock, which is obtained by looking at the maximum constant out of all the guards of the automaton. We will first restrict to only M-abstractions. The case of LU-abstractions, different bounds for each clock, and bounds obtained by static analysis are addressed in the end of this section, where by adding a simple gadget to this example, all the cases are taken care of.

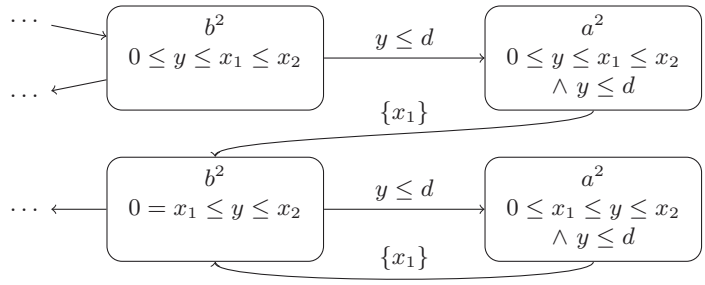


Figure 7.2: Part of $ZG(\mathcal{V}_2)$

A small gadget illustrating the idea

Consider $\mathcal{V}_k$ and $\mathcal{W}_k$ from Figure 7.1 and let us say that $k = 2$. Starting at the state b^2 of $\mathcal{V}_2$ with the zone $0 \le y \le x_1 \le x_2$, there are two reachable zones with state b^2. This is depicted in Figure 7.2 where after two traversals of the cycle formed by b^2 and a^2, we reach a zone that is invariant for the cycle. Moreover, from the two zones with state b^2 in Figure 7.2, reseting x_1 followed by y as $\mathcal{R}_1$ (in Figure 7.4) does, we reach the same zone $0 \le y \le x_1 \le x_2$.

In contrast starting in b^2 of $\mathcal{W}_2 = SNZ(\mathcal{V}_2)$ from $0 \le y \le x_1 \le x_2 \le z$ gives at least d zones. The part of $ZG(\mathcal{W}_2)$ in Figure 7.3 gives the sequence of transitions in the zone graph of $\mathcal{W}_2$ starting from the zone $(b^2, 0 \le y \le x_1 \le x_2 \le z)$ by successive iterations of the cycle that goes through b^2, a_1^2 and a_2^2. After a certain point, every traversal induces an extra distance between the clocks y and z. Clearly, there are at least d zones in this case. Resetting x_1 followed by y as $\mathcal{R}_1$ (in Figure 7.4) does still yields d zones. Note that when M is the bound function associating the maximum constant d to every clock and when we consider the M-abstractions, the abstracted zones still maintain this distance. It is not difficult to add extra dummy transitions to get constants that maintain this phenomenon for LU-abstractions and other refined ways of obtaining bound functions. We will address this issue in the end of this section.

Automaton exhibiting the blowup

We now exploit this situation observed in $\mathcal{V}_2$ and $\mathcal{W}_2$ to give an example of a TBA $\mathcal{A}_n$ whose zone graph has a number of zones linear in the number of clocks, but $\mathcal{B}_n = SNZ(\mathcal{A}_n)$ has a zone graph of size exponential in the number of clocks.

Automaton $\mathcal{A}_n$, in Figure 7.5, is constructed from the automata gadgets $\mathcal{V}_k$ and $\mathcal{R}_k$ as shown in Figures 7.1 and 7.4. Observe that the role of $\mathcal{R}_k$ is to enforce an order $0 \le y \le x_1 \le \cdots \le x_k$ between clock values. By induction on k one can compute that there are only two zones at locations b^k since

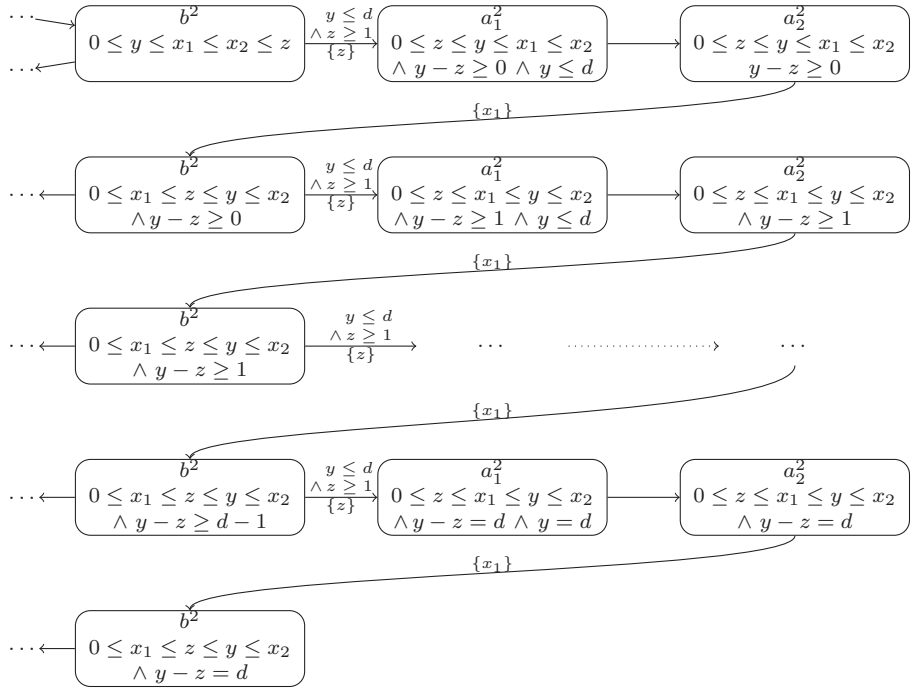


Figure 7.3: Part of $ZG(\mathcal{W}_2)$.

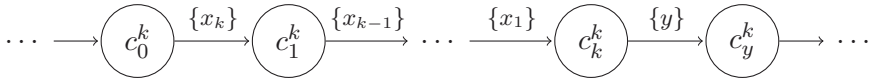


Figure 7.4: The gadget $\mathcal{R}_k$.

Figure 7.5: Automata $\mathcal{A}_n$ (left) and $\mathcal{B}_n = SNZ(\mathcal{A}_n)$ (right).

$\mathcal{R}_{k+1}$ made the two zones in b^{k+1} collapse into the same zone in b^k. Hence the number of nodes in the zone graph of $\mathcal{A}_n$ is $\mathcal{O}(n)$.

Let us now consider $\mathcal{B}_n$, the strongly non-Zeno automaton obtained from $\mathcal{A}_n$ following [TYB05]. Every gadget $\mathcal{V}_k$ gets transformed to $\mathcal{W}_k$ as shown in Figure 7.5. While exploring $\mathcal{W}_k$, one introduces a distance between the clocks x_{k-1} and x_k. So when leaving it one gets zones with $x_k - x_{k-1} \geq c$, where $c \in \{0, 1, 2, \ldots, d\}$. The distance between x_k and x_{k-1} is preserved by $\mathcal{R}_{k-1}$. In consequence, $\mathcal{W}_n$ produces at least $d+1$ zones. For each of these zones $\mathcal{W}_{n-1}$ produces $d+1$ more zones. In the end, the zone graph of $\mathcal{B}_n$ has at least $(d+1)^{n-k+1}$ zones at the state b^k. The zones obtained with the state b^k are of the form

$$0 \leq x_1 = \ldots = x_{k-1} \leq z \leq y \leq x_k \leq \ldots \leq x_n$$
$$\wedge \bigwedge_{i \in \{k,\ldots,n-1\}} x_{i+1} - x_i \geq c_i \qquad \text{where each } c_i \in \{0, 1, \ldots, d\}$$

So the zone graph has at least $(d+1)^{n-1}$ zones at state b^2. Hence, the zone graph of $\mathcal{B}_n$ contains at least $(d+1)^{n-1}$ zones.

We have thus shown that zone graph of $\mathcal{A}_n$ has $\mathcal{O}(n)$ zones while that of $\mathcal{B}_n = SNZ(\mathcal{A}_n)$ has an exponential number of zones even when the constant d is 1. One could argue that the transformation in [TYB05] can be transformed in such a way to prevent the combinatorial explosion. In particular, it is often suggested to replace $z \geq 1$ by a guard that matches the biggest constant in the automaton, that is $z \geq d$ in our case. However, this would still yield an exponential blowup as every zone with state b^k yields two different zones with state b^{k-1} that do not collapse going through $\mathcal{R}_{k-1}$. Observe also that the construction shows that even with two clocks the number of zones blows exponentially in the binary representation of d. Note that the automaton $\mathcal{A}_n$ does not have a non-Zeno accepting run. Hence, every search algorithm is compelled to explore all the zones of $\mathcal{B}_n$.

The case of general bound functions

For the above example, we considered a simple maximum bounds function. A small modification will give the same constant d for every clock and in fact even the LU-abstractions will boil down to the M-abstraction thanks to this modification. Consider the gadget $\mathcal{V}_{dummy}$ shown in Figure 7.6. From every state of $\mathcal{A}_n$ draw a transition to the state q of this gadget. This ensures that every clock has the bound d in every state. As this includes both lower and upper bounded guards, the LU abstractions are identical to the M-abstractions. In fact, even static analysis gives the same bounds.

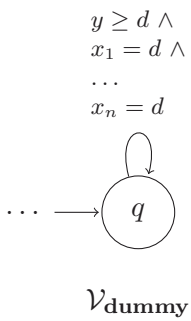


Figure 7.6: Gadget to be added for LU-abstractions and bounds by static analysis

7.2 A new construction

We have seen that the strongly non-Zeno construction, that modifies the syntax of the automaton, could potentially give rise to an exponential blowup. Our solution stems from a realization that we only need one non-Zeno run satisfying the Büchi condition and so in a way transforming an automaton to strongly non-Zeno is excessive. We propose not to modify the automaton, but to introduce additional information to the simulation graph. As we are considering only convex abstractions in this part of the thesis, the nodes in the simulation graph would indeed consist only of zones.

Remark 7.2.1 Recall Remark 6.6.1 about terminology. As we consider only convex abstractions, we will write *abstract zone graph* to mean a simulation graph. Additionally, we will write $ZG^{\mathfrak{a}}(\mathcal{A})$ instead of $SG^{\mathfrak{a}}(\mathcal{A})$.

We aim to decide if a given path in an abstract zone graph has a non-Zeno instantiation. We first check when could all instantiations of a path be Zeno. There are essentially two reasons for this:

- *blocking clocks:* there may be clocks x that are reset finitely many times but bound infinitely many times by guards $x \leq c$:

$$\bullet \xrightarrow{x\leq 1} \bullet \xrightarrow{\{x\}} \bullet \to \cdots \underbrace{\to \bullet \xrightarrow{x\leq 2} \bullet \to \bullet \xrightarrow{x\leq 1} \bullet \to \cdots}_{\text{suffix with no reset of } x}$$

- *zero checks:* time may not be able to elapse at all due to infinitely many transitions that check $x = 0$, forcing x to stay at 0:

$$\bullet \xrightarrow{\{y\}} \bullet \xrightarrow{x=0} \bullet \xrightarrow{\{x\}} \bullet \xrightarrow{y=0} \bullet \xrightarrow{\{y\}} \bullet \xrightarrow{x=0} \cdots$$

The task is to find if there exists an infinite run in $ZG^{\mathfrak{a}}(\mathcal{A})$ that does not have blocking clocks and zero-checks that prevent time elapse. The method that we propose tackles these two problems as follows. Blocking clocks are

handled by first detecting a maximal strongly connected component (SCC) of the abstract zone graph and repeatedly discarding the transitions that bound some blocking clock until a non-trivial SCC with no such clocks is obtained. This algorithm runs in time polynomial in the size of the zone graph for every abstraction. For zero checks, we introduce a *guessing zone graph* construction to detect nodes where time can elapse.

Guessing zone graph $GZG^{\mathfrak{a}}(\mathcal{A})$

The necessary and sufficient condition for time elapse in a node in spite of zero-checks is to have every reachable zero-check from that node preceded by a corresponding reset.

Figure 7.7: Time can elapse in the node $\surd$

Therefore, the aim is to check if there exists a node (q, Z) in $ZG^{\mathfrak{a}}(\mathcal{A})$ such that there is a path from (q, Z) back to itself in which every zero-check is preceded by a corresponding reset. This would instantiate to an infinite run of $\mathcal{A}$ that can elapse time despite the zero-checks. This is what the guessing zone graph construction achieves.

The nodes will now be triples (q, Z, Y) where $Y \subseteq X$ is the set of clocks that can potentially be equal to 0. It means in particular that other clock variables, i.e. those from $X - Y$ are assumed to be bigger than 0. We write $(X - Y) > 0$ for the constraint saying that all the variables in $X - Y$ are not 0. The role of Y sets will become obvious in the construction below. In short, from a node $(q, Z, \emptyset)$, that is with $Y = \emptyset$, every reachable zero-check will be preceded by the reset of the variable that is checked, and hence nothing prevents a time elapse in this node.

Definition 7.2.2 Let $\mathcal{A}$ be a TBA over a set of clocks X. The *guessing zone graph* $GZG^{\mathfrak{a}}(\mathcal{A})$ has nodes of the form (q, Z, Y) where (q, Z) is a node in $ZG^{\mathfrak{a}}(\mathcal{A})$ and $Y \subseteq X$. The initial node is (q_0, Z_0, X), with (q_0, Z_0) the initial node of $ZG^{\mathfrak{a}}(\mathcal{A})$. In $GZG^{\mathfrak{a}}(\mathcal{A})$ there are transitions:

- $(q, Z, Y) \Rightarrow^{t}_{\mathfrak{a}} (q', Z', Y \cup R)$ if there is a transition $(q, Z) \Rightarrow^{t}_{\mathfrak{a}} (q', Z')$ in $ZG^{\mathfrak{a}}(\mathcal{A})$ with $t = (q, g, R, q')$, and there are valuations $v \in Z$, $v' \in Z'$, and $\delta \in \mathbb{R}_{\geq 0}$ such that $v + \delta \vDash (X - Y) > 0$ and $(q, v) \xrightarrow{\delta, t} (q', v')$;
- $(q, Z, Y) \Rightarrow^{\tau}_{\mathfrak{a}} (q, Z, Y')$, on a new auxiliary letter τ, for $Y' = \emptyset$ or $Y' = Y$.

The additional component Y expresses some information about possible valuations with which we can take a transition. The first case is about

transitions that are realizable when clocks outside Y are positive. While it is formulated in a more general way, one can think of this transition as being instantaneous: $\delta = 0$. Then we have the second kind of transitions, namely the transitions on τ, that allow us to nondeterministically guess when time can pass.

It will be useful to distinguish some nodes and transitions of $GZG^{\mathfrak{a}}(\mathcal{A})$. We begin by formally defining the notion of zero-check.

Definition 7.2.3 (Zero-check) We call a transition (q, g, R, q') a *zero-check* if the guard g implies $x = 0$ for some clock x. We lift this definition to $GZG^{\mathfrak{a}}(\mathcal{A})$ to say that a transition $(q, Z, Y) \xrightarrow{(q,g,R,q')} (q', Z', Y')$ of $GZG^{\mathfrak{a}}(\mathcal{A})$ is a zero-check if (q, g, R, q') is a zero-check. This implies that for all $v \in Z$, and all $\delta \in \mathbb{R}_{\geq 0}$ such that $v + \delta \vDash g$ we have $(v + \delta)(x) = 0$.

We will be particularly interested in the following types of nodes to find non-Zeno accepting runs.

Definition 7.2.4 A node (q, Z, Y) of $GZG^{\mathfrak{a}}(\mathcal{A})$ is *clear* if the third component is empty: $Y = \emptyset$. A node is *accepting* if q is an accepting state.

Example 7.2.5 Figure 7.8 depicts a TBA $\mathcal{A}_1$ along with its zone graph $ZG^{\mathfrak{a}}(\mathcal{A}_1)$ and its guessing zone graph $GZG^{\mathfrak{a}}(\mathcal{A}_1)$ where τ-loops have been omitted.

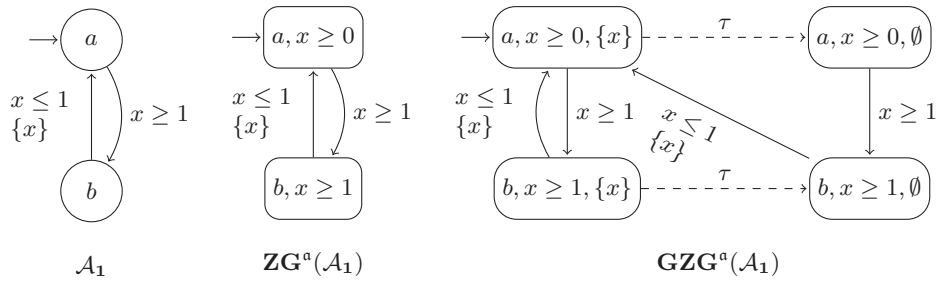


Figure 7.8: A TBA $\mathcal{A}_1$ and the guessing zone graph $GZG^{\mathfrak{a}}(\mathcal{A}_1)$ (with τ self-loops omitted for clarity).

Notice that directly from the definition it follows that a path in $GZG^{\mathfrak{a}}(\mathcal{A})$ determines a path in $ZG^{\mathfrak{a}}(\mathcal{A})$ obtained by removing τ transitions and the third component from nodes.

In order to state the main theorem succinctly we need some extra notions.

Definition 7.2.6 A variable x is *bounded* by a transition (q, g, R, q') if the guard g implies $x \leq c$ for some constant c. This definition can be lifted to $GZG^{\mathfrak{a}}(\mathcal{A})$: variable x is bounded by the $(q, Z, Y) \xrightarrow{(q,g,R,q')} (q', Z', Y')$ if it is bounded by (q, g, R, q'). This would imply that for all $v \in Z$ and $\delta \in \mathbb{R}_{\geq 0}$

such that $v + \delta \vDash g$, we have $(v + \delta)(x) \leq c$ for some $c \in \mathbb{N}$. A variable is *reset* by the transition if it belongs to the reset set R of the transition.

Definition 7.2.7 We say that a path is *blocked* if there is a variable that is bounded infinitely often and reset only finitely often by the transitions on the path. Otherwise the path is called *unblocked*.

Obviously, paths corresponding to non-Zeno runs are unblocked.

Theorem 7.2.8 *A TBA $\mathcal{A}$ has a non-Zeno run satisfying the Büchi condition iff there exists an unblocked path in $GZG^{\mathfrak{a}}(\mathcal{A})$ visiting both an accepting node and a clear node infinitely often.*

The proof of Theorem 7.2.8 follows from Lemmas 7.2.9 and 7.2.10 below. It is in Lemma 7.2.10 that the third component of states is used. In Section 6.2 we had recalled the progress criterion [AD94] stated in Definition 6.2.2 that characterizes the paths in region graphs that have non-Zeno instantiations. We had mentioned that it cannot be directly extended to zone graphs since their transitions are not pre-stable. Lemma 7.2.10 below shows that by slightly complicating the zone graph we can recover a result very similar to Lemma 4.13 in [AD94].

Lemma 7.2.9 If $\mathcal{A}$ has a non-Zeno run satisfying the Büchi condition, then in $GZG^{\mathfrak{a}}(\mathcal{A})$ there is an unblocked path visiting both an accepting node and a clear node infinitely often.

Proof
Let ρ be a non-Zeno run of $\mathcal{A}$:

$$(q_0, v_0) \xrightarrow{\delta_0, t_0} (q_1, v_1) \xrightarrow{\delta_1, t_1} \cdots$$

By Theorem 6.3.3, it is a concretization of a path σ in $ZG^{\mathfrak{a}}(\mathcal{A})$:

$$(q_0, Z_0) \Rightarrow_{\mathfrak{a}}^{t_0} (q_1, Z_1) \Rightarrow_{\mathfrak{a}}^{t_1} \cdots$$

Let σ' be the following sequence:

$$(q_0, Z_0, Y_0) \Rightarrow_{\mathfrak{a}}^{\tau} (q_0, Z_0, Y_0') \Rightarrow_{\mathfrak{a}}^{t_0} (q_1, Z_1, Y_1) \Rightarrow_{\mathfrak{a}}^{\tau} (q_1, Z_1, Y_1') \Rightarrow_{\mathfrak{a}}^{t_1} \cdots$$

where $Y_0 = X$, Y_i is determined by the transition, and $Y_i' = Y_i$ unless $\delta_i > 0$ when we put $Y_i' = \emptyset$. We need to see that this is indeed a path in $GZG^{\mathfrak{a}}(\mathcal{A})$. For this we need to see that every transition $(q_i, Z_i, Y_i') \Rightarrow_{\mathfrak{a}}^{t_i} (q_{i+1}, Z_{i+1}, Y_{i+1})$ is realizable from a valuation v such that $v \vDash (X - Y_i') > 0$. But an easy induction on i shows that actually $v_i \vDash (X - Y_i') > 0$.

Since ρ is non-Zeno there are infinitely many i with $Y_i' = \emptyset$. Since the initial run is non-Zeno, σ' is unblocked.

□

Lemma 7.2.10 Suppose $GZG^{\mathfrak{a}}(\mathcal{A})$ has an unblocked path visiting infinitely often both a clear node and an accepting node then $\mathcal{A}$ has a non-Zeno run satisfying the Büchi condition.

Proof
Let σ be a path in $GZG^{\mathfrak{a}}(\mathcal{A})$ as required by the assumptions of the lemma (without loss of generality we assume every alternate transition is a τ transition):

$$(q_0, Z_0, Y_0) \Rightarrow_{\mathfrak{a}}^{\tau} (q_0, Z_0, Y_0') \Rightarrow_{\mathfrak{a}}^{t_0} \cdots (q_i, Z_i, Y_i) \Rightarrow_{\mathfrak{a}}^{\tau} (q_i, Z_i, Y_i') \Rightarrow_{\mathfrak{a}}^{t_i} \cdots$$

Take a corresponding path in $ZG^{\mathfrak{a}}(\mathcal{A})$ and one instantiation $\rho = (q_0, v_0), (q_1, v_1) \ldots$ that exists by Theorem 6.3.3. If it is non-Zeno then we are done.

Suppose ρ is Zeno. We now show how to build a non-Zeno instantiation of σ from ρ. Let X^r be the set of variables reset infinitely often on σ. As σ is unblocked, every variable not in X^r is bounded only finitely often. Since ρ is Zeno, there is an index m such that the duration of the suffix of the run starting from (q_m, v_m) is bounded by $1/2$, and no transition in this suffix bounds a variable outside X^r. Let $n > m$ be such that every variable from X^r is reset between m and n. Observe that $v_n(x) < 1/2$ for every $x \in X^r$.

Take positions i, j such that $i, j > n$, $Y_i = Y_j = \emptyset$ and all the variables from X^r are reset between i and j. We look at the part of the run ρ:

$$(q_i, v_i) \xrightarrow{\delta_i, t_i} (q_{i+1}, v_{i+1}) \xrightarrow{\delta_{i+1}, t_{i+1}} \ldots (q_j, v_j)$$

and claim that for every $\zeta \in \mathbb{R}_{\geq 0}$ the sequence of the form

$$(q_i, v_i') \xrightarrow{\delta_i, t_i} (q_{i+1}, v_{i+1}') \xrightarrow{\delta_{i+1}, t_{i+1}} \ldots (q_j, v_j')$$

is a part of a run of $\mathcal{A}$ where v_k' for $k = i, \ldots, j$ satisfy:

1. $v_k'(x) = v_k(x) + \zeta + 1/2$ for all $x \notin X^r$,
2. $v_k'(x) = v_k(x) + 1/2$ if $x \in X^r$ and x has not been reset between i and k.
3. $v_k'(x) = v_k(x)$ otherwise, i.e., when $x \in X^r$ and x has been reset between i and k.

Before proving this claim, let us explain how to use it to conclude the proof. The claim shows that in (q_i, v_i) we can pass $1/2$ units of time and then construct a part of the run of $\mathcal{A}$ arriving at (q_j, v_j') where $v_j'(x) = v_j(x)$ for all variables in X^r, and $v_j'(x) = v_j(x) + 1/2$ for other variables. Now, we can find $l > j$, so that the pair (j, l) has the same properties as (i, j). We can pass $1/2$ units of time in j and repeat the above construction getting a

longer run that has passed 1/2 units of time twice. This way we construct a run that passes 1/2 units of time infinitely often, hence it is non-Zeno. By the construction it passes also infinitely often through accepting nodes.

It remains to prove the claim. Take a transition $(q_k, v_k) \xrightarrow{\delta_k, t_k} (q_{k+1}, v_{k+1})$ and show that $(q_k, v'_k) \xrightarrow{\delta_k, t_k} (q_{k+1}, v'_{k+1})$ is also a transition allowed by the automaton. Let g and R be the guard of t_k and the reset of t_k, respectively.

First we need to show that $v'_k + \delta_k$ satisfies the guard of t_k. For this, we need to check if for every variable $x \in X$ the constraints in g concerning x are satisfied. We have three cases:

- If $x \notin X^r$ then x is not bounded by the transition t_k, that means that in g the constraints on x are of the form $(x > c)$ or $(x \geq c)$. Since $(v_k + \delta_k)(x)$ satisfies these constraints so does $(v'_k + \delta_k)(x) \geq (v_k + \delta_k)(x)$.

- If $x \in X^r$ and it is reset between i and k then $v'_k(x) = v_k(x)$ so we are done.

- Otherwise, we observe that $x \notin Y_k$. This is because $Y_i = \emptyset$, and then only variables that are reset are added to Y. Since x is not reset between i and k, it cannot be in Y_k. By definition of transitions in $GZG^{\mathfrak{a}}(\mathcal{A})$ this means that $g \wedge (x > 0)$ is consistent. We have that $0 \leq (v_k + \delta_k)(x) < 1/2$ and $1/2 \leq (v'_k + \delta_k)(x) < 1$. So $v'_k + \delta_k$ satisfies all the constraints in g concerning x as $v_k + \delta_k$ does.

This shows that there is a transition $(q_k, v'_k) \xrightarrow{\delta_k, t_k} (q_{k+1}, v')$ for the uniquely determined $v' = [R](v'_k + \delta_k)$. It is enough to show that $v' = v'_{k+1}$. For variables not in X^r it is clear as they are not reset. For variables that have been reset between i and k this is also clear as they have the same values in v'_{k+1} and v'. For the remaining variables, if a variable is not reset by the transition t_k then its value is the same in v' and v'_k. If it is reset then its value in v' becomes 0; but so it is in v'_{k+1}, and so the third condition holds. This proves the claim. □

Examples

Figure 7.8 depicts a TBA $\mathcal{A}_1$ along with $ZG^{\mathfrak{a}}(\mathcal{A}_1)$ and $GZG^{\mathfrak{a}}(\mathcal{A}_1)$ (where the τ-loops have been omitted). In order to fire transition $b \xrightarrow{x \leq 1, \{x\}} a$ time must not elapse in b. The third component Y does not help to detect that time cannot elapse in b as in $GZG^{\mathfrak{a}}(\mathcal{A}_1)$ the transition is allowed for both $Y = \{x\}$ and $Y = \emptyset$. However, as soon as a strongly-connected component (SCC) contains a transition $x \geq 1$ and a transition that resets x, it has a non-Zeno run, and the third component does not play any role.

The third component is only useful for the case where an SCC contains a zero-check for some clock x that is also reset on some transition in the SCC. In such a case, zero-checks may prevent time to elapse. We illustrate this case on the next two examples that emphasize how the third component added to the states of the zone graph allows to distinguish between Zeno runs and non-Zeno runs.

The TBA $\mathcal{A}_2$ shown in Figure 7.9 has only runs where the time cannot elapse at all. This is detected in $GZG^{\mathfrak{a}}(\mathcal{A}_2)$ as all states in the only non-trivial SCC have $Y = \{x, y\}$ as the third component. This means that from every state there exists a reachable zero-check that is not preceded by the corresponding reset, hence preventing time to elapse. Notice that the correctness of this argument relies on the fact that for every (q, Z, Y) in $GZG^{\mathfrak{a}}(\mathcal{A}_2)$, and for every transition $t = (q, g, R, q')$, even if t is fireable in $ZG^{\mathfrak{a}}(\mathcal{A}_2)$ from (q, Z), it must also be fireable *under the supplementary hypothesis* $(X - Y) > 0$ given by Y in $GZG^{\mathfrak{a}}(\mathcal{A}_2)$.

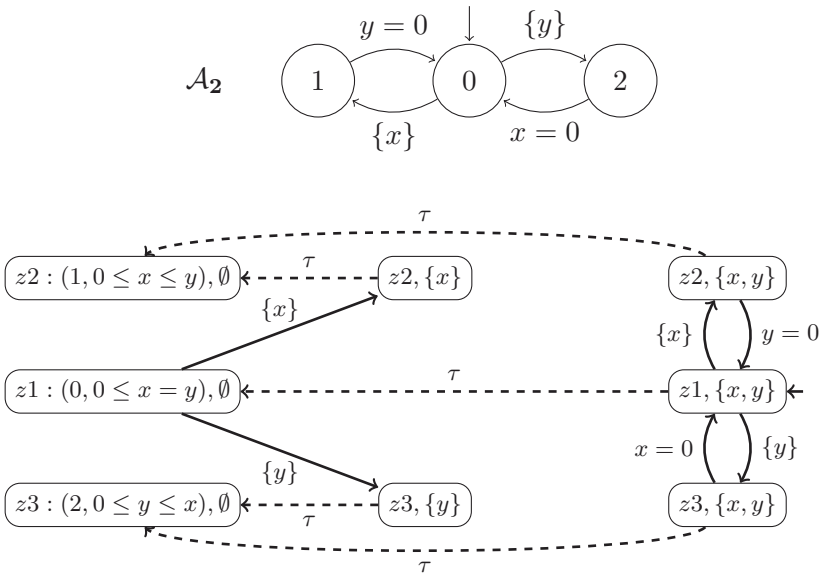
The TBA $\mathcal{A}_3$ in Figure 7.9 admits a non-Zeno run. This can be read from $GZG^{\mathfrak{a}}(\mathcal{A}_3)$ since the SCC composed of the four zones with $Y = \{x, y\}$ together with $(z_2, \emptyset)$ and $(z_3, \{y\})$ contains a clear node. This is precisely the state where time can elapse as every reachable zero-check is preceded by the corresponding reset.

The guessing zone graph construction therefore provides a way to infer existence of non-Zeno runs directly from the abstract zone graph. However, the solution requires adding an extra component Y which is a set of clocks. If the Y sets are added arbitrarily, then this solution causes an exponential blowup too. In the next section, we will see that if the abstractions satisfy a certain criterion then we need to add only a polynomial number of Y sets for each node of the zone graph.

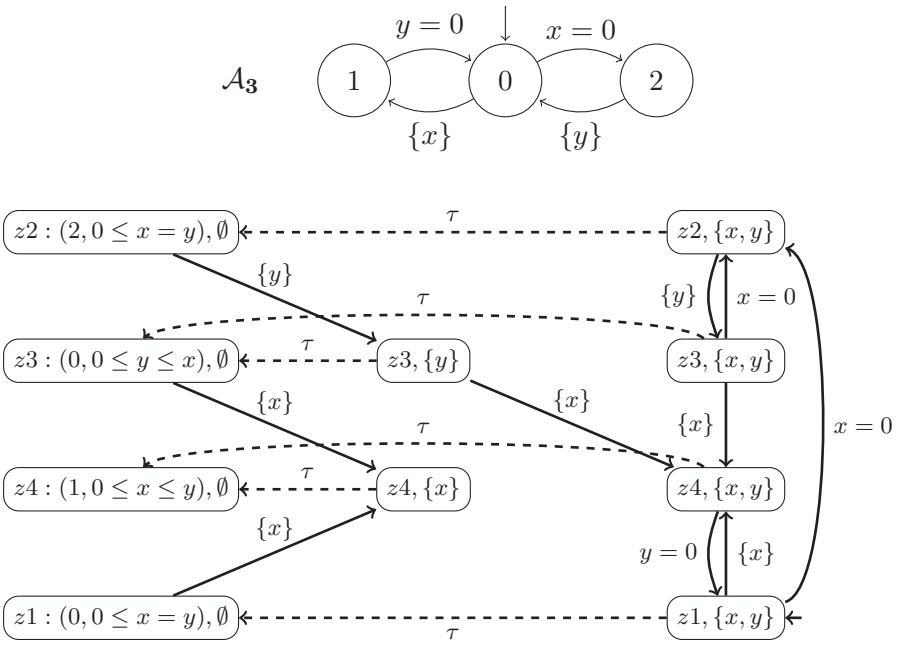
7.3 Complexity

We provide an explanation as to why the proposed solution does not produce an exponential blowup. At first it may seem that we have gained nothing because when adding arbitrary sets Y we have automatically caused exponential blowup to the zone graph. We will first consider an easy case: abstraction $Extra_M$ and the bound function M assigns the same maximum constant to every clock. In this case, we will show that the reachable part of the guessing zone graph from the initial node is still polynomial in the size of the original zone graph. This analysis would help us understand the crucial point that makes the construction efficient for $Extra_M$. With this gained knowledge, we will prune the construction later on in this section and characterize the abstractions that maintain polynomial complexity.

Remark 7.3.1 (Notation) We denote by $ZG^M(\mathcal{A})$ the abstract zone graph obtained using $Extra_M$. Similarly for $GZG^M(\mathcal{A})$. We denote by $\Rightarrow_M$



Reachable part of $\mathbf{GZG^a}(\mathcal{A}_2)$



Reachable part of $\mathbf{GZG^a}(\mathcal{A}_3)$

Figure 7.9: Examples of guessing zone graphs (τ self-loops have been omitted for clarity)

the abstract symbolic transition induced by $Extra_M$. We assume that M is a bound function that associates the same constant $d \geq 0$ to every clock.

We say that a zone *orders clocks* if for every two clocks x, y, the zone implies that at least one of $x \leq y$, or $y \leq x$ holds.

Lemma 7.3.2 Let Z be a zone and let M be a bound function that assigns the same constant to every clock. Then, if Z orders clocks, $Extra_M(Z)$ orders clocks.

Proof
Suppose for clocks x, y, we have $Z \vDash y \leq x$. Therefore we have $y - x \leq c$ for some $c \leq 0$ as one of the constraints defining Z. By the definition of $Extra_M$ abstraction as given in Definition 2.5.4, the constraint $y - x \leq c$ is removed by the abstraction only if $c > M_y$ and is changed to $y - x < -M_x$ if $-c > M_x$. As we have seen that $c \leq 0$, the former case is not possible. Hence in $Extra_M(Z)$, we still have $y - x \leq 0$ and so $y \leq x$. □

Lemma 7.3.3 If a node with a zone Z is reachable from the initial node of the zone graph $ZG^M(\mathcal{A})$ then Z orders clocks. The same holds for $GZG^M(\mathcal{A})$.

Proof
First notice that in the initial zone, all the clocks are equal to each other. Now, consider a zone Z that orders clocks. Let $(q, Z) \Rightarrow^t_M (q', Z')$ be a transition of $ZG^M(\mathcal{A})$. This means that there exists a transition $(q, Z) \Rightarrow^t (q', Z'_1)$ in the (unabstracted) zone graph $ZG(\mathcal{A})$ such that $Z' = Extra_M(Z'_1)$. Directly from the definition of transitions we have that Z'_1 orders clocks. It remains to check that, the clock ordering in Z'_1 is preserved in $Z' = Extra_M(Z'_1)$. This is true by Lemma 7.3.2. For the second statement observe that for every node (q, Z, Y) in $GZG^M(\mathcal{A})$, (q, Z) is reachable in $ZG^M(\mathcal{A})$. □

Suppose that Z orders clocks. We say that a set of clocks Y *respects the order given by* Z if whenever ($y \in Y$ and Z implies $x \leq y$) then $x \in Y$. In other words, Y is downward closed with respect to the ordering constraint in Z.

Lemma 7.3.4 If a node (q, Z, Y) is reachable from the initial node of the guessing zone graph $GZG^M(\mathcal{A})$ then Y respects the order given by Z.

Proof
The proof is by induction on the length of a path. In the initial node (q_0, Z_0, X), the set X obviously respects the order as it is the set of all clocks. Now take a transition $(q, Z, Y) \Rightarrow^t_M (q', Z', Y')$ with Y respecting the order in Z. We need to show that Y' respects the order in Z'. By the

definition of transitions in $GZG^M(\mathcal{A})$ there are $v \in Z$, $v' \in Z'$ and $\delta \in \mathbb{R}_{\geq 0}$ such that $(q, v) \xrightarrow{\delta,t} (q', v')$ and $v+\delta \vDash (X-Y) > 0$. Take $y \in Y'$ and suppose that Z' implies $x \leq y$ for some clock x. There are three cases depending on which of the variables y, x are being reset by the transition.

- If x is reset by the transition then, by definition $x \in Y'$.
- If y is reset and Z' implies $x \leq y$, then x must be checked for 0 on t. Hence $x \in Y$ and $x \in Y'$.
- The remaining case is when none of the two variables is reset by the transition. As $v' \in Z'$, we have that $v' \vDash x \leq y$; and in consequence $v \vDash x \leq y$. Since Z orders clocks and $v \in Z$, we must have that Z implies $x \leq y$. As y has not been reset, $y \in Y$. By assumption that Y orders clocks, $x \in Y$. Hence $x \in Y'$.

□

The above two lemmas give us the desired bound.

Theorem 7.3.5 *Let $|ZG^M(\mathcal{A})|$ be the size of the zone graph, and $|X|$ be the number of clocks in $\mathcal{A}$. The number of reachable nodes of $GZG^M(\mathcal{A})$ is bounded by $|ZG^M(\mathcal{A})|.(|X|+1)$.*

The theorem follows directly from the above two lemmas. Of course, imposing that zones have ordered clocks in the definition of $GZG^M(\mathcal{A})$ we would get the same bound for the entire $GZG^M(\mathcal{A})$.

Generalization

We have seen above that the number of guess sets for every node (q, Z) reachable in $ZG^M(\mathcal{A})$ is bounded by $|X|+1$ when the bound function M is a special one associating the same constant to every clock. The case of the general bound function M and other abstractions was not considered. The same construction does not give polynomial complexity in the general case. We first optimize this construction, considering an arbitrary abstraction and a general bound function that could associate different constants to different clocks. The results of this section also hold for the case of bounds obtained by static analysis (Section 2.6) where there is a bound function for each state of the automaton.

We prune the construction to obtain what is called the *reduced guessing zone graph*. The reduced guessing zone graph is a slight modification that restricts the guess sets to a particular set of clocks. A clock that is never checked for zero need not be remembered in sets Y. We restrict Y sets to only contain clocks that can indeed be checked for zero and we show that this is sound and complete for non-Zenoness.

We say that a clock x is *relevant* if it is checked for zero in the automaton, that is, if there exists a guard $x \leq 0$ or $x = 0$ in the automaton. We denote the set of relevant clocks by $\mathrm{Rl}(\mathcal{A})$. For a zone Z, let $\mathcal{C}_0(Z)$ denote the set of clocks x such that there exists a valuation $v \in Z$ with $v(x) = 0$. The clocks that can be checked for zero before being reset in a path from (q, Z), lie in $\mathrm{Rl}(\mathcal{A}) \cap \mathcal{C}_0(Z)$.

Definition 7.3.6 (Reduced guessing zone graph) Let $\mathcal{A}$ be a timed automaton with clocks X. The *reduced guessing zone graph* $rGZG^{\mathfrak{a}}(\mathcal{A})$ has nodes of the form (q, Z, Y) where (q, Z) is a node in $ZG^{\mathfrak{a}}(\mathcal{A})$ and $Y \subseteq \mathrm{Rl}(\mathcal{A}) \cap \mathcal{C}_0(Z)$. The initial node is $(q_0, Z_0, \mathrm{Rl}(\mathcal{A}))$, with (q_0, Z_0) the initial node of $ZG^{\mathfrak{a}}(\mathcal{A})$. For $t = (q, g, R, q')$, there is a transition $(q, Z, Y) \Rightarrow^t_{\mathfrak{a}} (q', Z', Y')$ with $Y' = (Y \cup R) \cap \mathrm{Rl}(\mathcal{A}) \cap \mathcal{C}_0(Z')$ if there is $(q, Z) \Rightarrow^t_{\mathfrak{a}} (q', Z')$ in $ZG^{\mathfrak{a}}(\mathcal{A})$ and some valuation $v \in Z$ such that $v \vDash (\mathrm{Rl}(\mathcal{A}) - Y) > 0$ and $v \vDash g$. A new auxiliary letter τ is introduced that adds transitions $(q, Z, Y) \Rightarrow^\tau_{\mathfrak{a}} (q, Z, Y')$ for $Y' = \emptyset$ or $Y' = Y$.

Observe that $rGZG^{\mathfrak{a}}(\mathcal{A})$ is the guessing zone graph $GZG^{\mathfrak{a}}(\mathcal{A})$ restricted to the clocks that are relevant and are equal to 0 in the respective zone, given by the set $\mathrm{Rl}(\mathcal{A}) \cap \mathcal{C}_0(Z)$. For example, in the automaton $\mathcal{A}_1$ of Figure 7.8, the set of relevant clocks $\mathrm{Rl}(\mathcal{A}_1)$ is empty.

We use the same notion of clear nodes and unblocked paths as in Definitions 7.2.4 and 7.2.7. We get the following theorem.

Theorem 7.3.7 *A timed automaton $\mathcal{A}$ has a non-Zeno run iff there exists an unblocked path in $rGZG^{\mathfrak{a}}(\mathcal{A})$ visiting a clear node infinitely often.*

The proof of Theorem 7.3.7 is in the same lines as for the guessing zone graph, from the following two lemmas.

Lemma 7.3.8 If $\mathcal{A}$ has a non-Zeno run, then in $rGZG^{\mathfrak{a}}(\mathcal{A})$ there is an unblocked path visiting a clear node infinitely often.

Proof
Let ρ be a non-Zeno run of $\mathcal{A}$:

$$(q_0, v_0) \xrightarrow{\delta_0, t_0} (q_1, v_1) \xrightarrow{\delta_1, t_1} \cdots$$

Since $\mathfrak{a}$ is complete, ρ is an instantiation of a path π in $ZG^{\mathfrak{a}}(\mathcal{A})$:

$$(q_0, Z_0) \Rightarrow^{t_0}_{\mathfrak{a}} (q_1, Z_1) \Rightarrow^{t_1}_{\mathfrak{a}} \cdots$$

Let σ be the following sequence of transitions:

$$(q_0, Z_0, Y_0) \Rightarrow^\tau_{\mathfrak{a}} (q_0, Z_0, Y_0') \Rightarrow^{t_0}_{\mathfrak{a}} (q_1, Z_1, Y_1) \Rightarrow^\tau_{\mathfrak{a}} (q_1, Z_1, Y_1') \Rightarrow^{t_1}_{\mathfrak{a}} \cdots$$

where $Y_0 = Rl(\mathcal{A})$, Y_i is determined by the transition relation in $rGZG^{\mathfrak{a}}(\mathcal{A})$, and $Y'_i = Y_i$ unless $\delta_i > 0$ when we put $Y'_i = \emptyset$. We need to see that σ is indeed a path in $rGZG^{\mathfrak{a}}(\mathcal{A})$. For this we need to see that every transition $(q_i, Z_i, Y'_i) \Rightarrow_{\mathfrak{a}}^{t_i} (q_{i+1}, Z_{i+1}, Y_{i+1})$ is realizable from a valuation $v \in Z_i$ such that both $v \vDash (Rl(\mathcal{A}) - Y'_i) > 0$ and $v \vDash g_i$ where g_i is the guard of t_i. We prove this by an induction on the run. As by the definition of ρ, $v_i + \delta_i \vDash g_i$ for all $i \geq 0$, we only need to prove that $v_i + \delta_i \vDash (\mathrm{Rl}(\mathcal{A}) - Y'_i) > 0$. This is clearly true for valuation $v_0 + \delta_0 \in Z_0$.

Assume that $v_i + \delta_i \vDash (\mathrm{Rl}(\mathcal{A}) - Y'_i) > 0$. We now prove that $v_{i+1} + \delta_{i+1} \vDash (\mathrm{Rl}(\mathcal{A}) - Y'_{i+1}) > 0$. Firstly, observe that $Y_{i+1} = (Y'_i \cup R_i) \cap \mathcal{C}_0(Z_{i+1}) \cap \mathrm{Rl}(\mathcal{A})$. Therefore a clock $x \in \mathrm{Rl}(\mathcal{A}) - Y_{i+1}$ either belongs to $\mathrm{Rl}(\mathcal{A}) - Y'_i$ in which case it is greater than 0 by induction hypothesis, or otherwise we have $x \in Y'_i$ but $x \notin \mathcal{C}_0(Z_{i+1})$. By the definition of $\mathcal{C}_0(Z_{i+1})$, all valuations $v \in Z_{i+1}$ satisfy $v(x) > 0$ and so in particular, $v_{i+1}(x) > 0$. This leads to $v_{i+1} \vDash (\mathrm{Rl}(\mathcal{A}) - Y_{i+1}) > 0$ which easily extends to $v_{i+1} + \delta_{i+1} \vDash (\mathrm{Rl}(\mathcal{A}) - Y'_{i+1}) > 0$.

Since ρ is non-Zeno there are infinitely many i with $Y'_i = \emptyset$. It is also straightforward to check that σ' is unblocked. □

Lemma 7.3.9 Suppose $rGZG^{\mathfrak{a}}(\mathcal{A})$ has an unblocked path visiting infinitely often a clear node then $\mathcal{A}$ has a non-Zeno run.

Proof

The proof follows the same lines as the proof of Lemma 7.2.10 with the additional information that for all clocks x that do not belong to $\mathrm{Rl}(\mathcal{A})$, we have $g \wedge (x > 0)$ consistent for all guards g.

Let $\pi : (q_0, Z_0, Y_0) \Rightarrow_{\mathfrak{a}}^{t_0} \ldots$ be the unblocked path of $rGZG^{\mathfrak{a}}(\mathcal{A})$ that visits a clear node infinitely often. Since $\mathfrak{a}$ is sound, take an instantiation $\rho : (q_0, v_0) \xrightarrow{\delta_0, t_0} \ldots$ of $\mathcal{A}$. If ρ is non-Zeno, we are done.

Suppose ρ is Zeno, there exists an index m such that all clocks $v_n(x) < 1/2$ for all $x \in X^r$ and for all $n \geq m$. Take indices $i, j \geq m$ such that $Y_i = Y_j = \emptyset$ and all clocks in X^r are reset between i and j. We look at the sequence $(q_i, v_i) \xrightarrow{\delta_i, t_i} \ldots (q_j, v_j)$ and claim that every sequence of the form

$$(q_i, v'_i) \xrightarrow{\delta_i, t_i} (q_{i+1}, v'_{i+1}) \xrightarrow{\delta_{i+1}, t_{i+1}} \ldots (q_j, v'_j)$$

is a part of a run of $\mathcal{A}$ provided there is $\zeta \in \mathbb{R}_{\geq 0}$ such that the following three conditions hold for all $k = i, \ldots, j$:

1. $v'_k(x) = v_k(x) + \zeta + 1/2$ for all $x \notin X^r$,
2. $v'_k(x) = v_k(x) + 1/2$ if $x \in X^r$ and x has not been reset between i and k.
3. $v'_k(x) = v_k(x)$ otherwise, i.e., when $x \in X^r$ and x has been reset between i and k.

It is easy to see that the run obtained by replacing every such $i - j$ interval of ρ by the above sequence gives a non-Zeno run, since a $1/2$ time unit has been elapsed infinitely often.

We now show that the above is indeed a valid run of $\mathcal{A}$. For this we need to first show that $v'_k + \delta_k$ satisfies the guard in t_k. Let g be the guard.

For $x \notin X^r$, from the assumption that ρ is unblocked, we know that g could only be of the form $x > c$ or $x \geq c$. So $v'_k(x)$ clearly satisfies g. If $x \in X^r$ and is reset between i and k, $v'_k(x) = v_k(x)$ and so we are done. Consider the case when $x \in X^r$ and is not reset between i and k. Observe that $x \notin Y_k$. This is because $Y_i = \emptyset$, and then only variables that are reset are added to Y. Since x is not reset between i and k, it cannot be in Y_k. By definition of transitions in $rGZG^{\mathfrak{a}}(\mathcal{A})$, if $x \in \mathrm{Rl}(\mathcal{A})$ this means that $g \wedge (x > 0)$ is consistent. But for $x \notin \mathrm{Rl}(\mathcal{A})$ by definition, $g \wedge (x > 0)$ is consistent. We have that $0 \leq (v_k + \delta_k)(x) < 1/2$ and $1/2 \leq (v'_k + \delta_k)(x) < 1$. So $v'_k + \delta_k$ satisfies all the constraints in g concerning x as $v_k + \delta_k$ does.

It can also be seen that the valuation obtained from v'_k by resetting the clocks in transition t_k is the valuation v'_{k+1}. □

Abstractions with polynomial complexity

We have seen in the simple case of $Extra_M$ and simple bounds that the number of Y sets that occur for every node is just $|X|+1$. This was possible due to the crucial fact that the zones obtained in reachable part of the zone graph ordered the clocks. The abstraction $Extra_M$ maintained the order when the M was a simple bound. In the reduced guessing zone graph case, we require the abstraction to maintain the order on a restricted set of clocks.

Definition 7.3.10 (Weakly order-preserving) An abstraction $\mathfrak{a}$ *weakly preserves orders* if for all clocks $x, y \in \mathrm{Rl}(\mathcal{A}) \cap \mathcal{C}_0(Z)$, $Z \vDash y \leq x$ iff $\mathfrak{a}(Z) \vDash y \leq x$.

Assume that $\mathfrak{a}$ weakly preserves orders, then for every reachable node (q, Z, Y) in $rGZG^{\mathfrak{a}}(\mathcal{A})$, the zone Z orders the clocks in $\mathrm{Rl}(\mathcal{A}) \cap \mathcal{C}_0(Z)$. We now show that Y is downward closed with respect to this order given by Z: for clocks $x, y \in \mathrm{Rl}(\mathcal{A}) \cap \mathcal{C}_0(Z)$, if $Z \vDash x \leq y$ and $y \in Y$, then $x \in Y$. This entails that there are at most $|\mathrm{Rl}(\mathcal{A})| + 1$ downward closed sets to consider, thus giving a polynomial complexity.

Theorem 7.3.11 *Let $\mathcal{A}$ be a timed automaton. If $\mathfrak{a}$ weakly preserves orders, then the reachable part of $rGZG^{\mathfrak{a}}(\mathcal{A})$ is $\mathcal{O}(|\mathrm{Rl}(\mathcal{A})|)$ bigger than the reachable part of $ZG^{\mathfrak{a}}(\mathcal{A})$.*

Proof
We prove by induction on the transitions in $rGZG^{\mathfrak{a}}(\mathcal{A})$ that for every reach-

able node (q, Z, Y) the set Y is downward closed with respect to Z on the clocks in $\mathrm{Rl}(\mathcal{A}) \cap \mathcal{C}_0(Z)$. This is true for the initial node $(q_0, Z_0, \mathrm{Rl}(\mathcal{A}))$.

Now, assume that this is true for (q, Z, Y). Take a transition $(q, Z, Y) \Rightarrow_{\mathfrak{a}}^{t} (q', Z', Y')$ with $t = (q, g, R, q')$. By definition, $Y' = (Y \cup R) \cap \mathrm{Rl}(\mathcal{A}) \cap \mathcal{C}_0(Z')$. Suppose $Z' \vDash x \leq y$ for some $x, y \in \mathrm{Rl}(\mathcal{A}) \cap \mathcal{C}_0(Z')$ and suppose $y \in Y'$. This could mean $y \in Y$ or $y \in R$. If $y \in R$, then x is zero-checked since $Z' \vDash x \leq y$. This gives $x \in Y$ and hence $x \in Y'$. If $y \notin R$ then we get $y \in Y$ and $Z \vDash x \leq y$. By hypothesis that Y is downward closed, $x \in Y$. In both cases $x \in Y'$. □

The following lemma shows that the M-extrapolations weakly preserve orders, for a general M.

Lemma 7.3.12 The abstractions Extra_M and Extra_M^+ weakly preserve orders.

Proof
For a zone Z, consider clocks $x, y \in \mathrm{Rl}(\mathcal{A}) \cap \mathcal{C}_0(Z)$ such that $Z \vDash y \leq x$. Clearly M_x and M_y are greater than or equal to 0 as they are relevant clocks and there are zero-checks involving these clocks. Therefore, by the same argument as in Lemma 7.3.2, we can get that $\mathit{Extra}_M(Z)$ satisfies $y \leq x$.

We now prove this for Extra_M^+. Firstly note that for a clock x in $\mathrm{Rl}(\mathcal{A})$ we have $M(x) \geq 0$. Moreover if $x \in \mathcal{C}_0(Z)$ we have that Z is consistent with $x \leq 0$. Hence, for a clock $x \in \mathrm{Rl}(\mathcal{A}) \cap \mathcal{C}_0(Z)$, Z is consistent with $x \leq M(x)$. Therefore, by definition, $\mathit{Extra}_M^+(Z)$ restricted to clocks in $\mathrm{Rl}(\mathcal{A}) \cap \mathcal{C}_0(Z)$ is identical to $\mathit{Extra}_M(Z)$ restricted to the same set of clocks. Since Extra_M weakly preserves orders, we get that Extra_M^+ weakly preserves orders too. □

The LU-extrapolations are not weakly order preserving. Let $x_1, \ldots, x_n$ be the set of clocks. Consider the zone Z_0 given by $x_1 = x_2 = \cdots = x_n$ and $x_1 \geq 0$. Now assume that all clocks are relevant due to guards of the form $x_i \leq 0$ for all $i \in \{1, \ldots, n\}$. However assume that there are no other guards in the automaton. So we have $U(x_i) = 0$ and $L(x_i) = -\infty$. With these bounds, $\mathit{Extra}_{LU}(Z_0) = \mathbb{R}_{\geq 0}$. The order information present in Z is lost in Extra_{LU} and hence the LU-extrapolations are not weakly order preserving. This shows that even the guessing zone graph construction could lead to an exponential blowup. We will show in the next section that this blowup is unavoidable.

7.4 NP-completeness for LU-abstractions

We have seen in the last section that our new construction for non-Zenoness might lead to an exponential blowup. Along with the observation of Section 7.1, we see that both the constructions for non-Zenoness discussed so

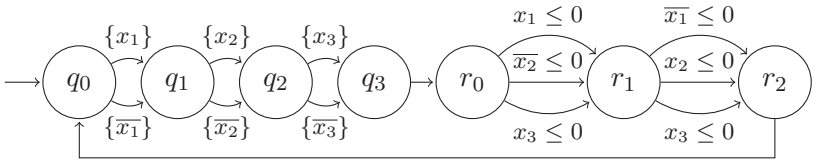


Figure 7.10: $\mathcal{A}_\phi^{NZ}$ for $\phi = (p_1 \vee \neg p_2 \vee p_3) \wedge (\neg p_1 \vee p_2 \vee p_3)$

far could potentially lead to an exponential blowup. This motivates us to study the complexity of the following decision problem:

Input	$\mathcal{A}$ and $ZG^{\mathfrak{a}}(\mathcal{A})$
Non-Zenoness problem ($\mathsf{NZP}^{\mathfrak{a}}$)	Does $\mathcal{A}$ have a non-Zeno run?

Thanks to Theorem 7.3.11 and Lemma 7.3.12, we know that $\mathsf{NZP}^{\mathfrak{a}}$ is polynomial for $\mathfrak{a}$ being one of the M-abstractions. Let us denote NZP^{LU} to be the non-Zenoness problem $\mathsf{NZP}^{\mathfrak{a}}$ when abstraction $\mathfrak{a}$ refers to either $Extra_{LU}$ or $Extra^+_{LU}$. We will now show that NZP^{LU} is NP-complete.

We give a reduction from the 3SAT problem: given a 3CNF formula ϕ, we build an automaton $\mathcal{A}_\phi^{NZ}$ that has a non-Zeno run iff ϕ is satisfiable. The size of the automaton will be linear in the size of ϕ. We will then show that the abstract zone graph $ZG^{LU}(\mathcal{A}_\phi^{NZ})$ is isomorphic to $\mathcal{A}_\phi^{NZ}$, thus completing the polynomial reduction from 3SAT to NZP^{LU}.

Automaton $\mathcal{A}_\phi^{NZ}$

Let $P = \{p_1, \ldots, p_k\}$ be a set of propositional variables and let $\phi = C_1 \wedge \cdots \wedge C_n$ be a 3CNF formula with n clauses. We define the timed automaton $\mathcal{A}_\phi^{NZ}$ as follows. Its set of clocks X equals $\{x_1, \ldots, x_k, \overline{x_1}, \ldots, \overline{x_k}\}$. For a literal λ, let $cl(\lambda)$ denote the clock x_i when $\lambda = p_i$ and the clock $\overline{x_i}$ when $\lambda = \neg p_i$. The set of states of $\mathcal{A}_\phi^{NZ}$ is $\{q_0, \ldots, q_k, r_0, \ldots, r_n\}$ where q_0 is the initial state. The transitions are as follows:

- for each p_i we have transitions $q_{i-1} \xrightarrow{\{x_i\}} q_i$ and $q_{i-1} \xrightarrow{\{\overline{x_i}\}} q_i$,
- for each clause $C_m = \lambda_1^m \vee \lambda_2^m \vee \lambda_3^m$, $m = 1, \ldots, n$, there are three transitions $r_{m-1} \xrightarrow{cl(\lambda) \leq 0} r_m$ where $\lambda \in \{\lambda_1^m, \lambda_2^m, \lambda_3^m\}$,
- transitions $q_k \to r_0$ and $r_n \to q_0$ with no guards and resets.

Figure 7.10 shows the automaton for the formula $(p_1 \vee \neg p_2 \vee p_3) \wedge (\neg p_1 \vee p_2 \vee p_3)$. Intuitively, a reset of x_i represents $p_i \mapsto true$ and a reset of $\overline{x_i}$ means $p_i \mapsto false$. From r_0 to r_2 we check if the formula is satisfied by this guessed assignment. This formula is satisfied by every assignment that maps p_3 to $true$. This can be seen from the automaton by picking a cycle

containing the transitions $q_2 \xrightarrow{\{x_3\}} q_3$, $r_0 \xrightarrow{x_3 \leq 0} r_1$ and $r_1 \xrightarrow{x_3 \leq 0} r_2$. On that path, time can elapse for instance in state q_0, since x_3 is reset before being zero-checked. Conversely, consider the assignment $p_1 \mapsto false$, $p_2 \mapsto true$ and $p_3 \mapsto false$ that does not satisfy the formula. Take a cycle that resets $\overline{x_1}$, x_2 and $\overline{x_3}$ corresponding to the assignment. Then none of the clocks that are checked for zero on the transitions from r_0 to r_1 has been reset. Notice that these transitions come from the first clause in the formula that evaluates to $false$ according to the assignment. To take a transition from r_0, one of x_1, $\overline{x_2}$ and x_3 must be zero and hence time cannot elapse.

Lemma 7.4.1 below states that if the formula is satisfiable, there exists a sequence of resets that allows time elapse in every loop. Conversely, if the formula is unsatisfiable, in every iteration of the loop, there is a zero-check that prevents time from elapsing.

Lemma 7.4.1 A 3CNF formula ϕ is satisfiable iff $\mathcal{A}_\phi^{NZ}$ has a non-Zeno run.

Proof
Let ϕ be a conjunction of n clauses $C_1, \ldots, C_n$. Assume that ϕ is satisfiable. Then, there exists a variable assignment $\chi : P \mapsto \{true, false\}$ that evaluates ϕ to true. This entails that in every clause C_m there is a literal λ_m that evaluates to true with χ.

We will now build a non-Zeno run ρ of $\mathcal{A}_\phi^{NZ}$ using this variable assignment χ. Let ρ be a run such that:

- from each configuration (q_{i-1}, v) for $i \in [1; k]$, ρ takes the transition $q_{i-1} \xrightarrow{\{x_i\}} q_i$ when $\chi(p_i) = true$ and the transition $q_{i-1} \xrightarrow{\{\overline{x_i}\}} q_i$ otherwise;
- from each configuration (r_{m-1}, v) for $m \in [1; n]$, ρ takes a transition $r_{m-1} \xrightarrow{cl(\lambda_m) \leq 0} r_m$ where λ_m is a literal evaluating to $true$ with respect to χ;
- and ρ lets 1 time unit elapse from each configuration with state r_n and moves to the state q_0; in all other states, there is no time elapse.

Note that as r_n occurs infinitely often, the run ρ is non-Zeno. It remains to prove that ρ is indeed a valid run of $\mathcal{A}_\phi^{NZ}$. For this, we need to prove that all zero-checked transitions can be crossed regardless of the unit time elapse. Consider the part of ρ between two successive configurations with state r_n.

$$\cdots (r_n, v) \xrightarrow{1} \cdots \xrightarrow{\{cl(\lambda_m)\}} \cdots (r_{m-1}, v'') \xrightarrow{cl(\lambda_m) \leq 0} (r_m, v'') \cdots (r_n, v') \xrightarrow{1} \cdots$$

By definition of ρ, λ_m is a literal that evaluates to $true$ according to χ. Hence, clock $cl(\lambda_m)$ is reset before in the corresponding $q_{j-1} \to q_j$ transition.

As $cl(\lambda_m)$ is reset and since ρ does not elapse time in states other than r_n, we have $v''(cl(\lambda_j^m)) = 0$. This permits the transition from r_{m-1} to r_m for all $m \in [1; n]$ and shows that the run ρ exists.

For the other direction, consider a non-Zeno run ρ of $\mathcal{A}_\phi^{NZ}$. Since ρ is non-Zeno, time elapses on infinitely many transitions in the run. Every infinite run of $\mathcal{A}_\phi^{NZ}$ visits a configuration with state r_n infinitely often. Consider two consecutive occurrences of r_n in ρ such that time elapses on some transition in the segment in between:

$$\cdots (r_n, v) \to \cdots (q_k, v') \to \cdots (r_{m-1}, v'') \xrightarrow{cl(\lambda_m) \leq 0} (r_m, v'') \cdots \to (r_n, v'') \cdots$$

By construction, for each $i \in [1; k]$ either x_i or $\overline{x_i}$ is reset on the segment from (r_n, v) to (q_k, v'). Let χ be the variable assignment that associates $true$ to p_i when x_i is reset, and $false$ otherwise, that is when $\overline{x_i}$ is reset. We prove that χ satisfies ϕ.

Consider the transition $(r_{m-1}, v'') \xrightarrow{cl(\lambda_m) \leq 0} (r_m, v'')$. For the transition to be enabled, we need to have $v''(cl(\lambda_m)) = 0$. Let $(q_{j-1}, v_{j-1}) \to (q_j, v_j)$ be the transition that resets either $cl(\lambda_m)$ or $cl(\overline{\lambda_m})$. Notice that time cannot elapse between (q_j, v_j) and (r_{m-1}, v''). So the time elapse should have occurred between (r_n, v) to (q_{j-1}, v_{j-1}). Thus it should be clock $cl(\lambda_m)$ that is reset in the transition $(q_{j-1}, v_{j-1}) \to (q_j, v_j)$. From the above definition of χ, we have λ_m evaluating to $true$ with χ and hence C_m evaluates to true with χ too. This holds for all the clauses. This shows that ϕ is satisfiable with χ being the satisfying assignment.

□

The NP-hardness of NZP^{LU} then follows due to the small size of the zone graph $ZG^{LU}(\mathcal{A}_\phi^{NZ})$.

Theorem 7.4.2 *The abstract zone graph $ZG^{LU}(\mathcal{A}_\phi^{NZ})$ is isomorphic to $\mathcal{A}_\phi^{NZ}$. The non-Zenoness problem $\mathsf{NZP}^{\mathsf{a}}$ is* NP*-complete for abstractions $Extra_{LU}$ and $Extra_{LU}^+$.*

Proof
We first prove that $ZG^{LU}(\mathcal{A}_\phi^{NZ})$ is isomorphic to $\mathcal{A}_\phi^{NZ}$. For every clock x, $L_x = -\infty$, hence $Extra_{LU}$ abstracts all the constraints $x_i - x_j \preccurlyeq_{ij} c_{ij}$ to $x_i - x_j < \infty$ except those of the form $x_0 - x_i \preccurlyeq_{0i} c_{0i}$ that are kept unchanged. Due to the guards in $\mathcal{A}_\phi^{NZ}$, for every reachable zone Z in $ZG(\mathcal{A}_\phi^{NZ})$ we have $x_0 - x_i \leq 0$ (i.e. $x_i \geq 0$). Therefore $Extra_{LU}(Z)$ is the zone defined by $\bigwedge_{x \in X} x \geq 0$ which is $\mathbb{R}_{\geq 0}^X$. For each state of $\mathcal{A}_\phi^{NZ}$, the zone $\mathbb{R}_{\geq 0}^X$ is the only reachable zone in $ZG^{LU}(\mathcal{A}_\phi^{NZ})$, hence showing the isomorphism. The result transfers to $Extra_{LU}^+$ as it is coarser than $Extra_{LU}$.

The NP-hardness of NZP^{LU} then follows from Lemma 7.4.1. The membership to NP will be proved in Lemma 7.4.3 in the next section. □

Notice that the type of zero checks in $\mathcal{A}_\phi^{NZ}$ is crucial to Theorem 7.4.2. Replacing zero-checks of the form $x \leq 0$ by $x = 0$ does not modify the semantics of $\mathcal{A}_\phi^{NZ}$. However, this yields $L_x = 0$ for every clock x. Hence, the constraints of the form $x_i - x_j \leq 0$ are not abstracted: $Extra_{LU}$ then preserves the ordering among the clocks. Each sequence of clock resets leading from q_0 to q_k yields a distinct ordering on the clocks. Thus, there are exponentially many LU-abstracted zones with state q_k. As a consequence, the polynomial reduction from 3SAT is lost. Indeed, when $L_x = 0$ for all clocks, the abstraction becomes weakly order preserving and hence Theorem 7.3.11 can be applied to give a polynomial algorithm.

Lemma 7.4.3 The abstractions $Extra_{LU}$ and $Extra^+_{LU}$ do not weakly preserve orders. The non-Zenoness problem $\mathsf{NZP}^{\mathfrak{a}}$ is in NP for $Extra_{LU}$ and $Extra^+_{LU}$.

Proof
The proof of Theorem 7.4.2 gives an example that illustrates that LU-extrapolations do not weakly preserve orders. This also holds for $Extra^+_{LU}$ as it is coarser than $Extra_{LU}$.

For the NP membership, let N be the number of nodes in $ZG^{LU}(\mathcal{A})$. Let us non-deterministically choose a node (q, Z). We assume that (q, Z) is reachable as this can be checked in polynomial time on $ZG^{LU}(\mathcal{A})$.

We augment (q, Z) with an empty guess set of clocks. From the node $(q, Z, \emptyset)$, we non-deterministically simulate a path π of the (non-reduced) guessing zone graph as in Definition 7.2.2. We avoid taking τ transitions on this path. This ensures that the guess sets accumulate all the resets on π. During the simulation, we also keep track of a separate set U containing all the clocks that are bounded from above on a transition in π.

We write $\Rightarrow_{\mathfrak{a}}^*$ to denote the transitive closure of $\Rightarrow_{\mathfrak{a}}$. If during the simulation one reaches a node (q, Z, Y) such that $U \subseteq Y$, then we have a cycle $(q, Z, \emptyset) \Rightarrow_{\mathfrak{a}}^* (q, Z, Y) \Rightarrow_{\mathfrak{a}}^\tau (q, Z, \emptyset)$ that is unblocked and that visits a clear node infinitely often. Also, since (q, Z) is reachable in $ZG^{LU}(\mathcal{A})$, (q, Z, X) is reachable in the guessing zone graph. Then $(q, Z, \emptyset)$ is reachable from (q, Z, X) with a τ transition. From Theorem 7.2.8 and from the fact that $Extra_{LU}$ and $Extra^+_{LU}$ are sound and complete [BBLP06] we get a non-Zeno run of $\mathcal{A}$.

Notice that it is sufficient to simulate $N \times (|X| + 1)$ transitions since we can avoid visiting a node (q', Z', Y') twice in π. □

7.4.1 Weakening the LU-abstractions

The LU-extrapolations do not weakly preserve orders in zones due to relevant clocks with $L_x = -\infty$ and $U_x \geq 0$. We show that this is the only

reason for NP-hardness. We slightly modify $Extra_{LU}$ to get an abstraction $Extra_{\overline{LU}}$ that is coarser than $Extra_M$, but it still weakly preserves orders.

Definition 7.4.4 (Weak L bounds) Let $\mathcal{A}$ be a timed automaton. Given the bounds L_x and U_x for every clock $x \in X$, the *weak lower bound* $\overline{L}$ is given by: $\overline{L}_x = 0$ if $x \in \mathrm{Rl}(\mathcal{A})$, $L_x = -\infty$ and $U_x \geq 0$, and $\overline{L}_x = L_x$ otherwise.

We denote $Extra_{\overline{LU}}$ the $Extra_{LU}$ abstraction obtained by choosing $\overline{L}$ instead of L. Notice that $Extra_{\overline{LU}}$ and $Extra_{LU}$ coincide when zero-checks are written $x = 0$ instead of $x \leq 0$ in the automaton. By definition of $Extra_{LU}$, we get the following.

Lemma 7.4.5 The abstraction $Extra_{\overline{LU}}$ weakly preserves orders.

$Extra_{\overline{LU}}$ coincides with $Extra_{LU}$ for a wide class of automata. For instance, when the automaton does not have a zero-check, $Extra_{\overline{LU}}$ is exactly $Extra_{LU}$, and the existence of a non-Zeno run can be decided in polynomial time. Similar to $Extra_{\overline{LU}}$ we can define $Extra^{+}_{\overline{LU}}$ which again weakly preserves orders.

The abstraction $Extra_{\overline{LU}}$ makes use of weak L bounds for every clock. If all clocks that are checked for $x \leq 0$ have a guard of the form $x \geq c$ then the weak abstraction coincides with $Extra_{LU}$. Notice that this is particularly the case for timed automata that do not have zero checks (i.e. guards like $x \leq 0$).

7.5 Concluding remarks

We have considered the question of detecting non-Zeno runs from an automaton. We have shown that the standard construction for non-Zenoness could cause a combinatorial blowup. We have proposed a solution avoiding this blowup for a class of abstractions weaker than LU-abstractions. For LU-abstractions, we show that a blowup is unavoidable unless P=NP.

We started the chapter with clearly distinguished goals (c.f. page 116). We will now summarize how we have achieved these goals.

1. The first goal was to examine the strongly non-Zeno construction of [TYB05], that has been recalled in Section 6.4. In Section 7.1, we have given an example of a gadget $\mathcal{V}_k$ and its strongly non-Zeno transformation $\mathcal{W}_k$ (Figure 7.1) which illustrate a property of this construction: the extra clock z allows to maintain distances. The zone graph of $\mathcal{V}_k$ has only four nodes (Figure 7.2), however the zone graph of $\mathcal{W}_k$ has d nodes (Figure 7.3) where $d \geq 1$ is a constant present in a guard of $\mathcal{V}_k$ and $\mathcal{W}_k$. We exploited this situation to give an automaton $\mathcal{A}_n$ with

$\mathcal{O}(n)$ zones in its zone graph, whereas its strongly non-Zeno transformation $\mathcal{B}_n$ has $\mathcal{O}((d+1)^n)$ zones.

2. We then proceeded to give a new solution which modifies the zone graph instead of the automaton. To this regard, we defined a guessing zone graph (Definition 7.2.2) and showed that this "inflated" zone graph could be used for checking Büchi non-emptiness (Theorem 7.2.8).

3. The following Section 7.3 dealt with analyzing the complexity of the guessing zone graph construction. We showed that the new construction induces only a polynomial blowup for zone graphs using the $Extra_M$ abstraction (Theorem 7.3.5). A scrupulous optimization of the guessing zone graph construction gave us the reduced guessing zone graph (Definition 7.3.6) which was again shown to be sound and complete for deciding non-Zenoness (Theorem 7.3.7). This reduced guessing zone graph maintains polynomial complexity for a class of abstractions called weakly order preserving (Theorem 7.3.11) and this includes the abstraction $Extra^+_M$ too. We noticed that the LU-extrapolations $Extra_{LU}$ and $Extra^+_{LU}$ are not weakly order preserving.

4. In Section 7.4.4, we show that deciding non-Zenoness from LU-abstract zone graphs is NP-complete (Theorem 7.4.2). We give a weakening of the LU-extrapolations that is weakly order preserving (Definition 7.4.4 and Lemma 7.4.5).

To summarize, in this chapter, we have proposed a new construction for non-Zenoness. We will use this construction to give an on-the-fly algorithm for the Büchi non-emptiness problem in the next chapter.

In Appendix B, we consider the decision problem:

Given an automaton $\mathcal{A}$, does it have a non-Zeno run?

Note the difference in the inputs to the problem from the decision problem $\mathsf{NZP}^{\mathfrak{a}}$ defined in page 135. In the above problem, the input is only $\mathcal{A}$ whereas in $\mathsf{NZP}^{\mathfrak{a}}$ the inputs were $\mathcal{A}$ and $ZG^{\mathfrak{a}}(\mathcal{A})$. We show that the above decision problem is PSPACE-complete. This implies that for inferring non-Zeno runs from an automaton, we need an object as complex as the zone graph. However, given our analysis of $\mathsf{NZP}^{\mathfrak{a}}$, it is surprising that the non-Zenoness question depends so heavily on abstractions that from a polynomial complexity to M-abstractions, we attain NP-completeness for LU-abstractions. This complexity analysis has also led to an interesting optimization for the reachability and liveness problems.

Recall the proof of NP-completeness of NZP^{LU} given in Theorem 7.4.2. For a 3CNF formula ϕ we built an automaton $\mathcal{A}_\phi$ that has a non-Zeno run

iff ϕ is satisfiable. The rest of the proof relied on the crucial fact that the zone graph $ZG^{LU}(\mathcal{A}_\phi)$ was isomorphic to $\mathcal{A}$. This was indeed possible as L_x was $-\infty$ for all x thanks to the guards of the form $x \leq 0$. Note that modifying $x \leq 0$ to $x = 0$ does not change the semantics of the automaton, but changes L_x to be 0 for all clocks. In this case, the zone graph $ZG^{LU}(\mathcal{A}_\phi)$ is no longer isomorphic to $\mathcal{A}_\phi$ and in fact it is exponentially larger than $\mathcal{A}_\phi$.

This gives us the easy optimization for analyzing an automaton $\mathcal{A}$ for reachability and liveness. Since both these algorithms go through the zone graph construction, changing all guards in $\mathcal{A}$ of the form $x = 0$ to $x \leq 0$ and removing all the guards $x \geq 0$ can produce a considerable gain. This modification has been incorporated in UPPAAL 4.1.5. Experimental results have shown a remarkable gain, in particular for timed Petri nets as the translation to timed automata may generate many guards like $x = 0$ and $x \geq 0$. For instance, checking reachability on a model of Fischer's protocol only explored 2541 nodes instead of 23042 nodes thanks to this optimization.

Chapter 8

Efficient Büchi emptiness

The Büchi non-emptiness problem for timed automata asks if there exists an accepting non-Zeno run in the automaton. We have seen in Section 6.3 that if there is an accepting run in the abstract zone graph[1] then there is an accepting run in the automaton. To detect non-Zenoness we have proposed a new construction in the previous chapter and analyzed its complexity for various abstractions.

In this chapter, we provide an on-the-fly algorithm for the Büchi non-emptiness problem using the guessing zone graph construction. Subsequently, we observe that in most cases, non-Zenoness could be detected directly from the standard abstract zone graph, without extra constructions. We provide an optimized on-the-fly algorithm taking into account these observations. We validate the gains of our approach by experimental results on some standard benchmarks.

Organization of the chapter

This chapter is divided into three sections.

- In Section 8.1, we give an on-the-fly algorithm for the Büchi non-emptiness problem by adapting Couvreur's on-the-fly algorithm for detecting strongly connected components in a graph [CDLP05].
- In Section 8.2, we provide an optimized on-the-fly algorithm that uses the guessing zone graph construction only when necessary.
- Finally, in Section 8.3, we include some experiments done with a prototype implementation of our algorithms.

[1] See Remark 6.6.1 for terminology

8.1 On-the-fly algorithm

We will use Theorem 7.2.8 to algorithmically check if an automaton $\mathcal{A}$ has a non-Zeno run satisfying the Büchi condition. The theorem requires to find an unblocked path in $GZG^{\mathfrak{a}}(\mathcal{A})$ visiting both an accepting node and a clear node infinitely often. This problem is similar to that of testing for emptiness of automata with generalized Büchi conditions as we need to satisfy two infinitary conditions at the same time. The requirement of a path being unblocked adds additional complexity to the problem. The best algorithms for testing emptiness of automata with generalized Büchi conditions are based on Tarjan's algorithm for strongly connected components (SCC) [SE05, GS09]. So this is the way we take here. In particular, we adopt the variant given by Couvreur [Cou99, CDLP05]. We have recalled the Couvreur's algorithm in Appendix C.

In general, the verification problem for timed systems involves checking if a network of timed automata $\mathcal{A}_1, \ldots, \mathcal{A}_n$ satisfies a given property ϕ. Assuming that ϕ can be translated into a (timed) Büchi automaton $\mathcal{A}_{\neg\phi}$, we reduce the verification problem to the emptiness of a timed Büchi automaton $\mathcal{A}$ defined as a product $\mathcal{A}_1 \times \mathcal{A}_2 \times \cdots \times \mathcal{A}_n \times \mathcal{A}_{\neg\phi}$ for some synchronization policy. Couvreur's algorithm is an extension of Tarjan's algorithm for computing maximal SCCs in a graph. One of its main features is that it stops as soon as an SCC with an accepting state has been found. In addition, it handles multiple accepting conditions efficiently. To this regard, the algorithm computes the set of accepting conditions in each visited SCC of $\mathcal{A}$. Initially, each state s in $\mathcal{A}$ is considered as a trivial SCC labelled with the accepting conditions of s. The algorithm computes the states of $\mathcal{A}$ on-the-fly in a depth-first search (DFS) manner starting from the initial state. During the search, when a cycle is found, all the SCCs in the cycle are merged into a bigger SCC Γ that inherits their accepting conditions. If Γ contains all the required accepting conditions, the algorithm stops declaring $\mathcal{A}$ to be not empty. Notice that Γ need not be maximal. Otherwise it resumes the DFS on $\mathcal{A}$. We direct the reader to Appendix C for further details on the Couvreur's algorithm.

We will now show how to enhance Couvreur's algorithm to detect runs that are not only accepting but also non-Zeno. It is achieved by associating extra information to the SCCs in $\mathcal{A}$. This information is updated when SCCs are merged similar to the method for accepting conditions.

Adapting Couvreur's algorithm

We apply Couvreur's algorithm for detecting SCCs in $GZG^{\mathfrak{a}}(\mathcal{A})$. During the computation of the SCCs, we keep track of whether an accepting node and a clear node have been seen. For the unblocked condition we use two sets of clocks U_Γ and R_Γ that respectively contain the clocks that are bounded from

above and the clocks that are reset in the SCC Γ. A clock from $U_\Gamma - R_\Gamma$ is called *blocking* since being bounded and not reset it puts a limit on the time that can pass on paths in Γ. At the end of the exploration of Γ we check if:

1. we have passed through an accepting node and a clear node,
2. there are no blocking clocks: $U_\Gamma \subseteq R_\Gamma$.

If the two conditions are satisfied then we can conclude saying that $\mathcal{A}$ has an accepting non-Zeno run. Indeed, a path passing infinitely often through all the nodes of Γ would satisfy the conditions of Theorem 7.2.8, giving a required run of $\mathcal{A}$. If the first condition does not hold then the same theorem says that Γ does not have a witness for a non-Zeno run of $\mathcal{A}$ satisfying the Büchi condition.

The interesting case is when the first condition holds but not the second. The following lemma yields an algorithm in that case.

Lemma 8.1.1 Let Γ be an SCC in $GZG^{\mathfrak{a}}(\mathcal{A})$ with an accepting node and a clear node, and such that $U_\Gamma \not\subseteq R_\Gamma$. There exists an unblocked path in Γ that visits both an accepting node and a clear node infinitely often iff there exists a sub-SCC $\Gamma' \subseteq \Gamma$ with an accepting node and a clear node and such that $U_{\Gamma'} \subseteq R_{\Gamma'}$.

Proof
Assume that Γ has an unblocked path that visits both an accepting node and a clear node infinitely often. Then, define Γ' as the set of nodes and edges that are visited infinitely often on that path.

Conversely, if such a sub-SCC Γ' exists, then consider an infinite path in Γ' that goes infinitely often through each node and each transition in Γ'. This path is unblocked and visits both an accepting node and a clear node. This path is also a path in Γ. □

We call *blocking edges* all the edges in Γ that upper-bound a clock from $U_\Gamma \setminus R_\Gamma$. We proceed as follows. We discard all the blocking edges from Γ as every unblocked path in Γ goes only finitely many times through these edges. In general, this yields several candidates for Γ'. Each of them is a proper sub-SCC of Γ. Then, we restart our algorithm on each such Γ'. Since we have discarded some edges from Γ (hence some resets), a clock may be now blocking in Γ'. If this is the case, the blocking edges in Γ' will be discarded, and the resulting sub-SCCs of Γ' will be explored, and so on. Observe that each transition in $GZG^{\mathfrak{a}}(\mathcal{A})$ will be visited at most $|X| + 1$ times, as we eliminate at least one clock at each restart. If after exploring the entire graph, the algorithm has not found a subgraph satisfying the two conditions then it declares that there is no run of $\mathcal{A}$ with the desired properties. The correctness of the procedure is based on Theorem 7.2.8. The entire procedure: exploring Γ, discarding blocking edges, exploring all

Γ' candidates, etc, can be done on-the-fly without storing Γ as described in [HS10].

The complexity of the algorithm given above follows from the linear complexity of Couvreur's algorithm and the remark about the bound on the number of times each transition is visited. We hence obtain the following.

Theorem 8.1.2 *The above algorithm is correct and runs in time* $\mathcal{O}(|GZG^{\mathfrak{a}}(\mathcal{A})| \cdot |X|)$.

By Theorem 7.3.5, we know that the size of the guessing zone graph $|GZG^{\mathfrak{a}}(\mathcal{A})|$ is $|ZG^{\mathfrak{a}}(\mathcal{A})| \cdot \mathcal{O}(|X|)$ for abstraction $Extra_M$. A similar result is obtained for abstraction $Extra^+_M$ when we consider the reduced guessing zone graph $rGZG^{\mathfrak{a}}(\mathcal{A})$ thanks to Theorem 7.3.11. However, for LU-abstractions, we know from Theorem 7.4.2 that the guessing zone graph could be $|ZG^{\mathfrak{a}}(\mathcal{A})| \cdot \mathcal{O}(2^{|X|})$. We will now see an optimized algorithm that runs well in practice even for LU-abstractions.

8.2 Optimizing use of non-Zenoness construction

Although the guessing zone graph provides a way to detect non-Zeno paths, it is useful only when the automaton indeed contains zero-checks. The next challenge therefore lies in optimizing the use of the guessing zone graph construction, that is, applying Couvreur's algorithm directly on the standard zone graph and using the guessing zone graph construction only when required.

The idea is to apply Couvreur's algorithm directly on $ZG^{\mathfrak{a}}(\mathcal{A})$ and find an SCC with an accepting node. An SCC is said to be *unblocked* if it contains no blocking clock; recall that it is a clock x that is checked for a guard which implies $x \leq c$ for a constant c and that is reset in no transition of the SCC.

Non-Zenoness can be ensured if the SCC in $ZG^{\mathfrak{a}}(\mathcal{A})$ satisfies one of the following conditions:

- It is unblocked and free from zero-checks.
- There is a clock x that is reset in the SCC and one of the transitions in the SCC implies $x \geq 1$.

For the second condition, note that such a reachable SCC instantiates into a path ρ of $\mathcal{A}$ whose suffix corresponds to repeated traversal of this SCC. Every traversal resets x and checks for a guard that implies $x \geq 1$. Therefore, at least 1 time unit elapses in each traversal, implying that ρ is a non-Zeno run. Notice that this relies on the same principle as the one used in the strongly non-Zeno construction [TYB05] (see Section 6.4). However, in our case we exploit the information from $\mathcal{A}$: we do not add any new clock. Our algorithm will compute on the fly the set L_Γ of clocks x such that $x \geq 1$

is implied by some guard in Γ. This is done in the same way as for U_Γ in the previous subsection. Then, Γ satisfies the second condition above if $L_\Gamma \cap R_\Gamma$ is not empty.

The first condition is justified by the following lemma.

Lemma 8.2.1 If $ZG^{\mathfrak{a}}(\mathcal{A})$ has an unblocked path that visits an accepting node infinitely often, and has only finitely many transitions with zero-checks, then $\mathcal{A}$ has a non-Zeno run satisfying the Büchi condition.

Proof
Let σ be the path in $ZG^{\mathfrak{a}}(\mathcal{A})$ as required by the assumptions of the lemma:

$$(q_0, Z_0) \Rightarrow_{\mathfrak{a}}^{t_0} \ldots (q_i, Z_i) \Rightarrow_{\mathfrak{a}}^{t_i} \ldots$$

Since zero-checks occur only finitely often in σ, we can find j such that the suffix $(q_j, Z_j) \Rightarrow_{\mathfrak{a}}^{t_j} \ldots$ of σ contains no zero-checks in its transitions. Let σ' be the following sequence:

$$(q_0, Z_0, Y_0) \Rightarrow_{\mathfrak{a}}^{\tau} (q_0, Z_0, Y_0') \Rightarrow_{\mathfrak{a}}^{t_0} (q_1, Z_1, Y_1) \Rightarrow_{\mathfrak{a}}^{\tau} (q_1, Z_1, Y_1') \Rightarrow_{\mathfrak{a}}^{t_1} \cdots$$

where $Y_0 = X$, Y_i is determined by the transition, and $Y_i' = Y_i$ for all $i \leq j$ and for $i > j$, $Y_i' = \emptyset$. Note that σ' is a path in $GZG^{\mathfrak{a}}(\mathcal{A})$. For this to be true, each transition $(q_i, Z_i, Y_i') \Rightarrow_{\mathfrak{a}}^{t_i} (q_{i+1}, Z_{i+1}, Y_{i+1})$ should be realizable from a valuation v_i such that $v_i \vDash (X - Y_i') > 0$. This is vacuously true if $i \leq j$ since $Y_i' = X$ for all $i \leq j$. For $i > j$, $Y_i' = \emptyset$ and since t_i does not contain a zero-check, the transition is realizable from a valuation v_i in which all clocks are strictly greater than 0.

Since σ is unblocked, σ' is unblocked too. By definition all but finitely many nodes for σ' are clear. Finally, σ' visits an accepting node infinitely often. By Theorem 7.2.8, $\mathcal{A}$ has a non-Zeno run satisfying the Büchi condition. □

The above two observations give a sufficient condition for terminating with a success when an SCC Γ with an accepting node is found in $ZG^{\mathfrak{a}}(\mathcal{A})$. If the above two conditions do not hold, then Γ has no clock bounded from below (i.e. $x \geq 1$) that is reset and Γ either has blocking clocks or zero-checks. If it has only blocking clocks, we apply the procedure that restarts the exploration with blocking edges removed, as described in Section 8.1. If Γ has zero-checks, we indeed use the guessing zone graph construction, however *restricted only to the nodes of* Γ. The problem is to know the initial set of clocks that need to be zero. We first define a few notations.

Let (q^Γ, Z^Γ) be a node of Γ. Let $GZG_{|\Gamma}(\mathcal{A})$ be the part of $GZG(\mathcal{A})$ rooted at (q^Γ, Z^Γ, X) and restricted only to the nodes and transitions that occur in Γ. We say that a run ρ of $\mathcal{A}$ is *trapped* in an SCC Γ of $ZG^{\mathfrak{a}}(\mathcal{A})$ if a suffix of ρ is an instantiation of a path in Γ. The following lemma justifies the use of the restricted guessing zone graph construction starting from (q^Γ, Z^Γ, X).

Lemma 8.2.2 The automaton $\mathcal{A}$ has an accepting non-Zeno run trapped in an SCC Γ of $ZG^{\mathfrak{a}}(\mathcal{A})$ iff $GZG_{|\Gamma}$ has an SCC that is accepting, unblocked and contains a clear node.

Proof
For the left-right direction, consider the following run ρ of $\mathcal{A}$ trapped in Γ:

$$(q_0, v_0) \xrightarrow{\delta_0, t_0} \dots (q_m, v_m) \xrightarrow{\delta_m, t_m} \dots$$

where $q_m = q^\Gamma$, $v_m \in Z^\Gamma$ and (q^Γ, Z^Γ) is a node of Γ. Consider the sequence σ':

$$(q_0, Z_0, Y_0) \Rightarrow_{\mathfrak{a}}^{\tau} (q_0, Z_0, Y_0') \Rightarrow_{\mathfrak{a}}^{t_0} (q_1, Z_1, Y_1) \Rightarrow_{\mathfrak{a}}^{\tau} (q_1, Z_1, Y_1') \Rightarrow_{\mathfrak{a}}^{t_1} \cdots$$

where

- (q_0, Z_0) is the initial node of $ZG^{\mathfrak{a}}(\mathcal{A})$, the zone Z_i is determined by the transition t_i,
- $Y_0 = X$, Y_i is determined by the transition,
- $Y_i' = Y_i$ for all $i \leq m$; for $i > m$, $Y_i' = \emptyset$ if $\delta_i > 0$ and $Y_i' = Y_i$ otherwise.

Observe that $Y_m = X$ and the suffix of σ' starting from (q_m, Z_m, Y_m) is a path of $GZG_{|\Gamma}(\mathcal{A})$. Since there are infinitely many i with $\delta_i > 0$, this suffix corresponds to an SCC that has a clear node. It is accepting and unblocked since the run ρ that we started with is accepting and non-Zeno.

For the right-left direction, note that an accepting, unblocked SCC with a clear node in $GZG_{|\Gamma}(\mathcal{A})$ corresponds to an accepting, unblocked path of $GZG(\mathcal{A})$ starting from (q^Γ, Z^Γ, X) that visits a clear node infinitely often. It is straightforward to see that (q^Γ, Z^Γ, X) is reachable from the initial node (q_0, Z_0, X) of $GZG(\mathcal{A})$ through a path in which for all transitions $(q, Z, Y) \xrightarrow{\tau} (q', Z', Y')$, $Y' = Y$. Indeed, the restriction of $GZG(\mathcal{A})$ to its nodes with $Y = X$ is isomorphic to the zone graph $ZG^{\mathfrak{a}}(\mathcal{A})$. From this path of $GZG(\mathcal{A})$ and using Lemma 7.2.10, we can construct a accepting, non-Zeno run of $\mathcal{A}$ that is trapped in Γ. □

Based on the above observations, we give the schema of the overall optimized algorithm in Figure 8.1. In the worst case, for $Extra_M$ abstraction, the algorithm runs in time $\mathcal{O}(|ZG^{\mathfrak{a}}(\mathcal{A})| \cdot |X|^2)$. When the automaton does not have zero-checks it runs in time $\mathcal{O}(|ZG^{\mathfrak{a}}(\mathcal{A})| \cdot |X|)$, irrespective of the abstraction used. When the automaton further has no blocking clocks, it runs in time $\mathcal{O}(|ZG^{\mathfrak{a}}(\mathcal{A})|)$, again, irrespective of the abstraction.

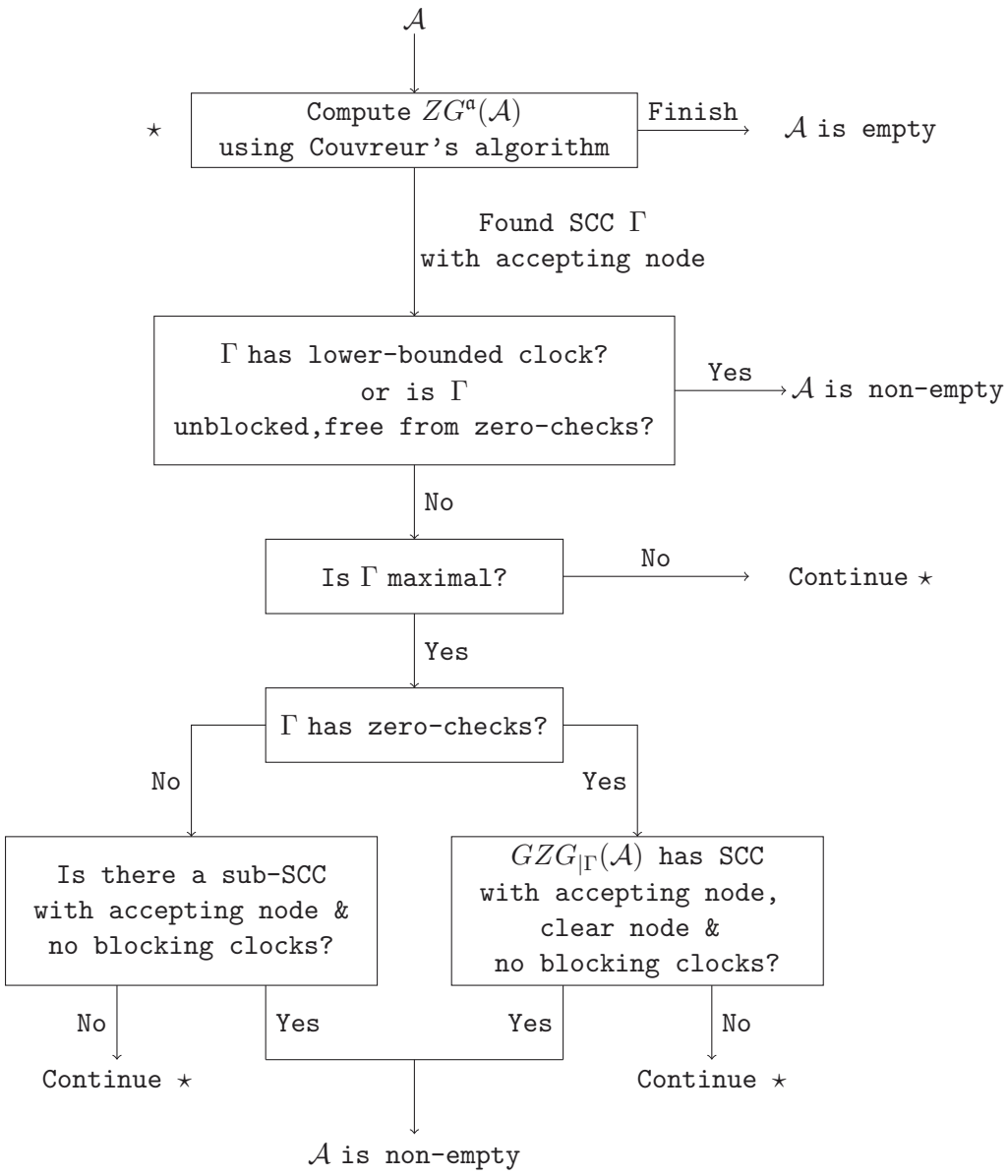


Figure 8.1: Algorithm to check for Büchi emptiness of $\mathcal{A}$. "Continue" loops back to computing $ZG^{\mathfrak{a}}(\mathcal{A})$ using Couvreur's Algorithm.

8.3 Experiments

We have implemented our algorithms in a prototype verification tool. We have considered only the abstraction $Extra_M$. The goal was to verify the effect of the guessing zone graph construction over the strongly non-Zeno construction. We see that the guessing zone graph construction applied naively induces a systematic polynomial increase in the zone graph. This

performs worse than the strongly non-Zeno in many cases. However, the optimized algorithm explained in the previous section performs substantially better than both the strongly non-Zeno and the naive guessing zone graph constructions. In many cases, we notice that non-Zenoness can be inferred from just the abstract zone graph itself.

Table 8.1 presents the results that we obtained on several classical examples. The "Models" column represents the product of the network of Timed Büchi Automata and the property to verify. We give the number of processes in the network for each model. A tick in the "Sat." columns tells that the property is satisfied by the model. The "Zone Graph" column gives the number of nodes in the zone graph (abstracted with $Extra_M$). Next, for the "Strongly non-Zeno" construction, we give the size of the resulting zone graph followed by the number of nodes that are visited during verification using the Couvreur's algorithm. Similarly for the "Guessing Zone Graph" but using the algorithm in section 8.1. Finally, the last column corresponds to our fully optimized algorithm as described in section 8.2.

We have considered three types of properties: reachability properties (mutual exclusion, collision detection for CSMA/CD), liveness properties (access to the resource infinitely often), and bounded response properties (which are reachability properties with real-time requirements). Reachability properties require to find a path to a target state starting from the initial state. Although this path is a finite sequence, it is realistic only if this finite sequence can be extended to a non-Zeno path of the automaton. Therefore, while verifying reachability properties, we check if the automaton has a non-Zeno path that contains the target state.

The strongly non-Zeno construction outperforms the guessing zone graph construction for reachability properties. This is particularly the case for mutual exclusion on the Fischer's protocol and collision detection for the CSMA/CD protocol. For liveness properties, the results are more balanced. On the one hand, the strongly non-Zeno construction is once again more efficient for the CSMA/CD protocol. On the other hand the differences are tight in the case of Fischer protocol. The guessing zone graph construction distinguishes itself for bounded response properties. Indeed, the Train-Gate model is an example of exponential blowup for the strongly non-Zeno construction.

We notice that on-the-fly algorithms perform well. Even when the graphs are big, particularly in case when automata are not empty, the algorithms are able to conclude after having explored only a small part of the graph. Our optimized algorithm outperforms the two others on most examples. Particularly, for the CSMA/CD protocol with 5 stations our algorithm needs to visit only 4841 nodes while the two other methods visited 8437 and 21038 nodes. This confirms our initial hypothesis: most of the time, the zone graph contains enough information to ensure time progress. As a consequence, checking non-Zenoness and emptiness is done at the same cost as

Models ($\mathcal{A}$)	Sat.	$ZG^{\mathfrak{a}}(\mathcal{A})$	$ZG^{M}(SNZ(\mathcal{A}))$		$GZG^{M}(\mathcal{A})$		Opt
		size	size	visited	size	visited	visited
Train-Gate2 (mutex)	√	134	194	194	400	400	134
Train-Gate2 (bound. resp.)		988	227482	352	3840	1137	292
Train-Gate2 (liveness)		100	217	35	298	53	33
Fischer3 (mutex)	√	1837	3859	3859	7292	7292	1837
Fischer4 (mutex)	√	46129	96913	96913	229058	229058	46129
Fischer3 (liveness)		1315	4962	52	5222	64	40
Fischer4 (liveness)		33577	147167	223	166778	331	207
FDDI3 (liveness)		508	1305	44	3654	79	42
FDDI5 (liveness)		6006	15030	90	67819	169	88
FDDI3 (bound. resp.)		6252	41746	59	52242	114	60
CSMA/CD4 (collision)	√	4253	7588	7588	20146	20146	4253
CSMA/CD5 (collision)	√	45527	80776	80776	260026	260026	45527
CSMA/CD4 (liveness)		3038	9576	1480	14388	3075	832
CSMA/CD5 (liveness)		32751	120166	8437	186744	21038	4841

Table 8.1: Experimental Results. The "Sat." column tells which properties are satisfied by the model. The "size" columns give the number of nodes in the corresponding graphs. The "visited" columns give the number of nodes that are visited by the corresponding algorithm. The results correspond to the Couvreur's algorithm for $ZG^{\mathfrak{a}}(SNZ(\mathcal{A}))$, the algorithm in Section 8.1 for $GZG(\mathcal{A})$ and the algorithm in Section 8.2 for the "Optimized" column.

checking emptiness only. This is in turn achieved at a cost that is similar to reachability checking.

Our optimization using lower bounds (that is $x \geq 1$) on clocks also proves useful for the FDDI protocol example. One of its processes has zero-checks, but since some other clock is bounded from below and reset, it was not necessary to explore the guessing zone graph to conclude non-emptiness.

8.4 Concluding remarks

In this chapter, we have given an on-the-fly algorithm for the Büchi non-emptiness problem using the guessing zone graph construction for non-Zenoness (Section 8.1). The solution proceeds by an adaptation of Couvreur's algorithm for detecting SCCs in a graph. The exact implementation details of Couvreur's algorithm have been deferred to Appendix C.

In the following Section 8.2, we provided an optimized use of the guessing zone graph construction. This optimized use lets us detect non-Zenoness directly from the zone graph in most cases. The optimized algorithm outperforms the strongly non-Zeno construction and the naive guessing zone graph by a substantial margin, as validated by experimental results in Section 8.3.

The experiments have been performed using the $Extra_M$ abstraction only. We plan to implement the reduced guessing zone graph construction (Definition 7.3.6) for LU-abstractions in a future implementation. As the initial experiments with $Extra_M$ are promising and in particular, in most

cases we infer non-Zenoness from the zone graph itself without extra constructions, our approach could be good in practice even for LU-abstractions, albeit the NP-completeness.

Chapter 9

Zenoness problem

Zeno runs portray unrealistic behaviours. It is therefore important to detect their existence before implementing timed automata. The current solution to this problem involves a check on the automaton syntax to ensure absence of Zeno runs (Section 6.5). The method is efficient as it is a static analysis, but it is not complete. We provide the first complete solution to this problem in this chapter.

We consider this syntactic check once again. It looks for loops in the automaton in which there is a clock that is reset and is bounded from below by some guard. The procedure suffers from being sufficient-only because it is applied on the automaton syntax. We consider the zone graph instead. A zone obtained after a transition $x \geq c$ maintains this information, as all valuations in this zone have the value of x bigger than c. If additionally, the abstraction of the zone maintains this information too, then one could get an efficient on-the-fly algorithm linear in the size of the abstract zone graph for the problem of detecting zeno runs.

We notice that the LU-abstractions do not always maintain the information about a clock x having a value greater than c in every valuation of the zone. The construction given above therefore cannot be used on the LU-abstract zone graph. It turns out that the problem of deciding Zeno runs given the automaton and the LU-abstract zone graph is NP-complete. However, a slight weakening of the LU-abstractions removes this problem of NP-completeness.

Organization of the chapter

In Section 9.1 we describe our new algorithm for Zenoness checking. It is called the slow zone graph construction. We follow up in Section 9.2 with the results on the Zenoness problem for LU-abstractions. We end the chapter with some concluding remarks in Section 9.3.

9.1 A new algorithm

We propose a new algorithm to detect existence of Zeno runs in an automaton. This new algorithm can be implemented on-the-fly. We begin with a definition.

Definition 9.1.1 (Lifting transitions) A transition $t = (q, g, R, q')$ is said to be *lifting* if the guard g implies $x \geq 1$ for some clock x.

The idea is to find if there exists a run of an automaton $\mathcal{A}$ in which every clock x that is reset infinitely often is lifted only finitely many times. In other words, after some point of time, all clocks that are reset are never lifted. This ensures that the automaton can take this run in a Zeno fashion. This amounts to checking if there exists a cycle in $ZG(\mathcal{A})$ where every clock that is reset is not lifted. Observe that when $(q, Z) \stackrel{x \geq c}{\Longrightarrow} (q', Z')$ is a transition of $ZG(\mathcal{A})$ that does not reset x, the target zone Z' entails $x \geq c$. Therefore, if a node (q, Z) is part of a cycle in the required form, then in particular, all the clocks that are greater than 1 in Z should not be reset in the cycle.

Based on the above intuition, our solution begins with computing the zone graph on-the-fly. At some node (q, Z) the algorithm nondeterministically guesses that this node is part of a cycle that yields a zeno run. This node transits to what we call the *slow mode*. In this mode, a reset of x in a transition is allowed from (q', Z') only if $Z' \wedge g$ is consistent with $x < 1$, where g is the guard of the transition. Note that $Z' \wedge g$ gives the set of valuations that actually cross the transition.

Before we define our construction formally, recall that we would be working with the abstract zone graph $ZG^{\mathfrak{a}}(\mathcal{A})$ and not $ZG(\mathcal{A})$. Therefore for our solution to work, the abstraction operator $\mathfrak{a}$ should remember the fact that a clock has a value greater than 1. For an automaton $\mathcal{A}$ over the set of clocks X, let $\mathrm{Lf}(\mathcal{A})$ denote the set of clocks that appear in a lifting transition of $\mathcal{A}$.

Definition 9.1.2 (Lift-safe abstractions) An abstraction $\mathfrak{a}$ is called *lift-safe* if for every zone Z and for every clock $x \in \mathrm{Lf}(\mathcal{A})$, $Z \models x \geq 1$ iff $\mathfrak{a}(Z) \models x \geq 1$.

We are now in a position to define our *slow zone graph* construction to decide if an automaton has a Zeno run.

Definition 9.1.3 (Slow zone graph) Let $\mathcal{A}$ be a timed automaton over the set of clocks X. Let $\mathfrak{a}$ be a lift-safe abstraction. The *slow zone graph* $\mathcal{S}ZG^{\mathfrak{a}}(\mathcal{A})$ has nodes of the form (q, Z, l) where $l \in \{\mathrm{free}, \mathrm{slow}\}$. The initial node is $(q_0, Z_0, \mathrm{free})$ where (q_0, Z_0) is the initial node of $ZG^{\mathfrak{a}}(\mathcal{A})$. For every transition $(q, Z) \Rightarrow^t_{\mathfrak{a}} (q', Z')$ in $ZG^{\mathfrak{a}}(\mathcal{A})$ with $t = (q, g, R, q')$, we have the following transitions in $\mathcal{S}ZG^{\mathfrak{a}}(\mathcal{A})$:

- a transition $(q, Z, \text{free}) \Rightarrow_{\mathfrak{a}}^{t} (q', Z', \text{free})$,
- a transition $(q, Z, \text{slow}) \Rightarrow_{\mathfrak{a}}^{t} (q', Z', \text{slow})$ if for all clocks $x \in R$, $Z \wedge g$ is consistent with $x < 1$,

A new letter τ is introduced that adds transitions $(q, Z, \text{free}) \Rightarrow_{\mathfrak{a}}^{\tau} (q, Z, \text{slow})$.

A node of the form (q, Z, slow) is said to be a *slow* node. A path of $\mathcal{SZG}^{\mathfrak{a}}(\mathcal{A})$ is said to be *slow* if it has a suffix consisting entirely of slow nodes. The τ-transitions take a node (q, Z) from the *free* mode to the *slow* mode. Note that the transitions of the slow mode are constrained further. A transition $(q, Z, \text{slow}) \Rightarrow_{\mathfrak{a}}^{t} (q', Z', \text{slow})$ can reset a clock x only when the set of valuations crossing the transition is consistent with $x < 1$. We say $Z \wedge g$ (and not just Z) should be consistent with $x < 1$. This is to cater to transitions that have both a reset of x and a guard of the form $x \geq 1$. So, even if the zone Z is consistent with $x < 1$, the valuations crossing the transition will have the clock value $x \geq 1$. In the slow mode, such transitions should not be allowed.

The correctness follows from the fact that there is a cycle in $\mathcal{SZG}^{\mathfrak{a}}(\mathcal{A})$ consisting entirely of slow nodes iff $\mathcal{A}$ has a Zeno run. This is detailed in the following two lemmas.

Lemma 9.1.4 If $\mathcal{A}$ has a Zeno run, then there exists an infinite slow path in $\mathcal{SZG}^{\mathfrak{a}}(\mathcal{A})$.

Proof
Let ρ be a Zeno run of $\mathcal{A}$:

$$(q_0, v_0) \xrightarrow{\delta_0, t_0} (q_1, v_1) \xrightarrow{\delta_1, t_1} \dots$$

Let π be the corresponding path in $ZG^{\mathfrak{a}}(\mathcal{A})$:

$$(q_0, Z_0) \Rightarrow_{\mathfrak{a}}^{t_0} (q_1, Z_1) \Rightarrow_{\mathfrak{a}}^{t_1} \dots$$

We construct an infinite slow path in $\mathcal{SZG}^{\mathfrak{a}}(\mathcal{A})$ from the path π. Let X^l be the set of clocks that are lifted infinitely often in π and let X^r be the set of clocks that are reset infinitely often in π. Let π^i denote the suffix of π starting from the position i.

Clearly, there exists an index m such that all the clocks that are lifted in π^m belong to X^l and the ones that are reset in π^m belong to X^r. Since ρ is Zeno, we have $X^l \cap X^r = \emptyset$. This shows that all the clocks that are reset in π^m are never lifted in its transitions. Let ρ^k denote the suffix of ρ starting from (q_k, v_k). There exists an index $k \geq m$ such that for all $j \geq k$, Z_j is consistent with $x < 1$ for all clocks $x \in X^r$ and we get the following path of $\mathcal{SZG}^{\mathfrak{a}}(\mathcal{A})$:

$$(q_0, Z_0, \text{free}) \Rightarrow_{\mathfrak{a}}^{t_0} \ldots \Rightarrow_{\mathfrak{a}}^{t_{j-1}} (q_j, Z_j, \text{free})$$
$$\Rightarrow_{\mathfrak{a}}^{\tau} (q_j, Z_j, \text{slow}) \Rightarrow_{\mathfrak{a}}^{t_j} (q_{j+1}, Z_{j+1}, \text{slow}) \Rightarrow_{\mathfrak{a}}^{t_{j+1}} \ldots$$

□

Lemma 9.1.5 If $\mathcal{SZG}^{\mathfrak{a}}(\mathcal{A})$ has an infinite slow path, then $\mathcal{A}$ has a Zeno run.

Proof
Let π be the slow path of $\mathcal{SZG}^{\mathfrak{a}}(\mathcal{A})$:

$$(q_0, Z_0, \text{free}) \Rightarrow_{\mathfrak{a}}^{t_0} \ldots \Rightarrow_{\mathfrak{a}}^{t_{j-1}} (q_j, Z_j, \text{free})$$
$$\Rightarrow_{\mathfrak{a}}^{\tau} (q_j, Z_j, \text{slow}) \Rightarrow_{\mathfrak{a}}^{t_j} (q_{j+1}, Z_{j+1}, \text{slow}) \Rightarrow_{\mathfrak{a}}^{t_{j+1}} \ldots$$

Take the corresponding path in $ZG^{\mathfrak{a}}(\mathcal{A})$ and an instance $\rho = (q_0, v_0) \xrightarrow{\delta_0, t_0} (q_1, v_1) \ldots$ which is a run of $\mathcal{A}$, as we have assumed that $\mathfrak{a}$ is a sound abstraction.

Let X^r be the set of clocks that are reset infinitely often and let X^l be the set of clocks that are lifted infinitely often in ρ. By the semantics of the slow mode and from our hypothesis of $\mathfrak{a}$ being lift-safe, after the index j, all clocks that are lifted once can never be reset again. Therefore, there exists an index $k \geq j$ such that the following hold:

- all clocks that are reset in ρ^k belong to X^r and all clocks that are lifted in a transition of ρ^k belong to X^l,
- for all $x \in X^l$ and for all $i \geq k$, $v_i(x) \geq c$ where c is the maximum constant appearing in a lifting transition of ρ^k.

We now modify the time delays of ρ^k to construct a run that elapses a bounded amount of time. Pick an increasing sequence of indices $i_1, i_2, \ldots$ in ρ^k such that $\delta_{i_m} > 0$, for all $m \in \mathbb{N}$. Define the new delays δ'_i for all $i \geq k$ as follows:

$$\delta'_i = \begin{cases} min(\delta_i, \frac{1}{2^j}) & \text{if } i = i_j \text{ for some } j \\ 0 & \text{otherwise} \end{cases}$$

Consider the run ρ' obtained by elapsing δ'_i time units after the index k:

$$(q_0, v_0) \xrightarrow{\delta_0, t_0} \ldots \xrightarrow{\delta_{k-1}, t_{k-1}} (q_k, v_k) \xrightarrow{\delta'_k, t_k} (q_{k+1}, v'_{k+1}) \xrightarrow{\delta'_{k+1}, t_{k+1}} \ldots$$

Clearly, ρ' is Zeno. It remains to prove that ρ' is a run of $\mathcal{A}$. Denote v_k by v'_k. We need to show that for all $i \geq k$, $v'_i + \delta'_i$ satisfies the guard in the

transition t_i. Call this guard g_i. Clearly, since $v'_i + \delta'_i \leq v_i + \delta_i$ by definition, if g_i is of form $x < c$ or $x \leq c$ then it is satisfied by the new valuation. Let us now consider the case when g_i is of the form $x \geq c$ or $x > c$. If $c \geq 1$, then we know that $x \in X^l$ from the assumption on k. But since $v_k(x) \geq c$ and x is not reset anywhere in ρ^k, $v'_i(x) \geq c$ for all i and hence the new valuation satisfies g_i. We are left with the case when g_i is of the form $x > 0$. However this follows since by definition of the new δ'_i, $v'_i + \delta'_i = 0$ iff $v_i + \delta_i = 0$. □

From the definition of $\mathcal{SZG}^{\mathfrak{a}}(\mathcal{A})$ it follows clearly that for each node (q, Z) of the zone graph there are two nodes in $\mathcal{SZG}^{\mathfrak{a}}(\mathcal{A})$: (q, Z, free) and (q, Z, slow). We thus get the following theorem.

Theorem 9.1.6 *Let* $\mathfrak{a}$ *be a lift-safe abstraction. The automaton* $\mathcal{A}$ *has a Zeno run iff* $\mathcal{SZG}^{\mathfrak{a}}(\mathcal{A})$ *has an infinite slow path. The number of reachable nodes of* $\mathcal{SZG}^{\mathfrak{a}}(\mathcal{A})$ *is at most twice the number of reachable nodes in* $ZG^{\mathfrak{a}}(\mathcal{A})$.

We now turn our attention towards some of the abstractions existing in the literature. We observe that both $Extra_M$ and $Extra^+_M$ are lift-safe and hence the Zenoness problem can be solved using the slow zone graph construction. However, in accordance with the announced NP-hardness of the problem for $Extra_{LU}$, we get that $Extra_{LU}$ is not lift-safe.

Lemma 9.1.7 The abstractions $Extra_M$ and $Extra^+_M$ are lift-safe.

Proof
Observe that for every clock that is lifted, the bound M is at least 1. It is now direct from the definitions that $Extra_M$ and $Extra^+_M$ are lift-safe. □

The LU-abstractions are not lift-safe. Consider an automaton in which there is a single clock x which is present in a guard $x \geq 1$. There are no other guards involving x, which gives $U_x = -\infty$. So for a zone $Z := x \geq 1$, by definition we will have $Extra_{LU}(Z) = Extra^+_{LU}(Z) = \mathbb{R}_{\geq 0}$, clearly showing that LU-extrapolations are not lift-safe.

9.2 NP-completeness for LU-extrapolations

We have seen that the LU-abstractions are not lift-safe and hence our new algorithm cannot be applied. This motivates us to consider the following decision problem:

Input	$\mathcal{A}$ and $ZG^{\mathfrak{a}}(\mathcal{A})$
Zenoness problem ($\mathsf{ZP}^{\mathfrak{a}}$)	Does $\mathcal{A}$ have a Zeno run?

By our slow zone graph construction of the previous section, this problem is polynomial for M-abstractions. As in the case of non-Zenoness (Section 7.4), this problem turns out to be NP-complete when the abstraction

operator $\mathfrak{a}$ is $Extra_{LU}$. We first show that the slow zone graph construction puts this problem in NP for LU-abstractions.

Lemma 9.2.1 The Zenoness problem $\mathsf{ZP}^{\mathfrak{a}}$ is in NP for abstractions $Extra_{LU}$ and $Extra^{+}_{LU}$.

Proof
We show the NP-membership using a technique similar to the slow zone graph construction. Let $ZG^{LU}(\mathcal{A})$ denote an abstract zone graph obtained using either $Extra_{LU}$ or $Extra^{+}_{LU}$. Since $Extra_{LU}$ is not lift-safe, the reachable zones in $ZG^{LU}(\mathcal{A})$ do not maintain the information about the clocks that have been lifted. Therefore, at some reachable zone (q, Z) we non-deterministically guess the set of clocks W that are allowed to be lifted in the future and go to a node (q, Z, W). From now on, there are transitions $(q, Z, W) \Rightarrow^{t}_{\mathfrak{a}} (q', Z', W)$ when:

- $(q, Z) \Rightarrow^{t}_{\mathfrak{a}} (q', Z')$ is a transition in $ZG^{LU}(\mathcal{A})$,
- if t contains a guard $x \geq c$ with $c \geq 1$, then $x \in W$,
- if t resets a clock x, then $x \notin W$

If a cycle is obtained that contains (q, Z, W), then the clocks that are reset and lifted in this cycle are disjoint and hence $\mathcal{A}$ has a Zeno run.

This shows that if $\mathcal{A}$ has a Zeno run we can non-deterministically choose a path of the above form and the length of this path is bounded by twice the number of zones in $ZG^{LU}(\mathcal{A})$ (which is our other input). This proves the NP-membership. □

For NP-hardness, similar to Section 7.4, we show a reduction from the 3SAT problem. Let $P = \{p_1, \ldots, p_k\}$ be a set of propositional variables. Let $\phi = C_1 \wedge \cdots \wedge C_n$ be a 3CNF formula with n clauses. Each clause C_m, $m = 1, 2, \ldots, n$ is a disjunction of three literals λ^m_1, λ^m_2 and λ^m_3. We construct in polynomial time an automaton $\mathcal{A}^Z_\phi$ and its zone graph $ZG^{LU}(\mathcal{A}^Z_\phi)$ such that $\mathcal{A}^Z_\phi$ has a Zeno run iff ϕ is satisfiable, thus proving the NP-hardness.

The automaton $\mathcal{A}^Z_\phi$ has clocks $\{x_1, \overline{x_1}, \ldots, x_k, \overline{x_k}\}$ with x_i and $\overline{x_i}$ corresponding to the literals p_i and $\neg p_i$ respectively. We denote the clock associated to a literal λ by $cl(\lambda)$. The set of states of $\mathcal{A}^Z_\phi$ is given by $\{q_0, q_1, \ldots, q_k\} \cup \{r_0, r_1, r_2, \ldots, r_n\}$ with q_0 being the initial state. The transitions are as follows:

- transitions $q_{i-1} \xrightarrow{\{x_i\}} q_i$ and $q_{i-1} \xrightarrow{\{\overline{x_i}\}} q_i$ for $i = 1, 2, \ldots, k$,
- a transition $q_k \to r_0$ with no guards and resets,
- for each clause C_m there are three transitions $r_{m-1} \xrightarrow{cl(\neg\lambda)\geq 1} r_m$ for each literal $\lambda \in \{\lambda^m_1, \lambda^m_2, \lambda^m_3\}$,

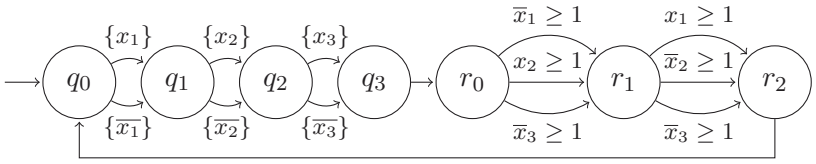


Figure 9.1: $\mathcal{A}^Z_\phi$ for $\phi = (p_1 \vee \neg p_2 \vee p_3) \wedge (\neg p_1 \vee p_2 \vee p_3)$

- a transition $r_n \to q_0$ with no guards and resets. This transition creates a cycle in $\mathcal{A}^Z_\phi$.

As an example, Figure 9.1 shows the automaton for the formula $(p_1 \vee \neg p_2 \vee p_3) \wedge (\neg p_1 \vee p_2 \vee p_3)$. Observe that the transitions $r_{m-1} \xrightarrow{cl(\neg\lambda)\geq 1} r_m$ check if the clock corresponding to the *negation* of λ is greater than 1. That is, $cl(\neg\lambda) = \overline{cl(\lambda)}$.

Clearly, $\mathcal{A}^Z_\phi$ can be constructed from ϕ in $\mathcal{O}(|\phi|)$ time. We now show that ϕ is satisfiable iff $\mathcal{A}^Z_\phi$ has a Zeno run.

Lemma 9.2.2 A 3CNF formula ϕ is satisfiable iff $\mathcal{A}^Z_\phi$ has a Zeno run.

Proof
For the left-to-right direction, suppose that ϕ is satisfiable. Then there exists a variable assignment $\chi : P \mapsto \{true, false\}$ that evaluates ϕ to true. We now build the Zeno run of $\mathcal{A}^Z_\phi$ using χ.

Pick an infinite run ρ of $\mathcal{A}^Z_\phi$. Clearly, it should have the following sequence of states repeated infinitely often:

$$q_0 \to \ldots q_k \to r_0 \to r_1 \to \ldots r_n \tag{9.1}$$

We choose the transitions for ρ that allow time elapse only by a finite amount. If $\chi(p_i) = true$, then we put $q_{i-1} \xrightarrow{\{x_i\}} q_i$ wherever $q_{i-1} \to q_i$ occurs in ρ. Otherwise $\chi(p_i) = false$ and we put $q_{i-1} \xrightarrow{\{\overline{x_i}\}} q_i$. We now need to choose the transitions $r_{m-1} \to r_m$ for $m = 1, \ldots, n$. Since χ is a satisfying assignment, every clause C_m has a literal λ that evaluates to true with χ. We choose the corresponding transition $r_{m-1} \xrightarrow{cl(\neg\lambda)\geq 1} r_m$. Observe that if λ evaluates to true, it implies that $cl(\lambda)$ was reset in one of the $q_i \to q_{i+1}$ transitions but not $cl(\neg\lambda)$.

Therefore, the above construction yields a sequence of transitions with the property that all clocks that are reset are never checked for greater than 1. This sequence can be taken by elapsing 1 time unit in the very first state, and then subsequently elapsing no time at all, thus giving a Zeno run in $\mathcal{A}^Z_\phi$.

We now prove the right-to-left direction. Let ρ be an infinite Zeno run of $\mathcal{A}^Z_\phi$. An infinite run should repeat the sequence of states given in (9.1).

Since ρ is Zeno, it has a suffix ρ^s such that for every clock x that is reset in ρ^s, $x \geq 1$ never occurs in the transitions of ρ^s. This is because if every suffix of ρ contains a clock that is both reset and checked for greater than 1, this would mean that there is a time elapse of one time unit occurring infinitely often, contradicting the hypothesis that ρ is Zeno.

Consider a segment $S = q_0 \to \dots q_n \to r_0 \to r_1 \to \dots r_k$ in ρ^s. We construct a satisfying assignment $\chi : P \mapsto \{true, false\}$ for ϕ from S.

- if S contains $q_{i-1} \xrightarrow{\{x_i\}} q_i$ then set $\chi(p_i) = true$
- otherwise, it implies that S contains $q_{i-1} \xrightarrow{\{\overline{x_i}\}} q_i$ in which case we set $\chi(p_i) = false$.

This shows that for a literal λ, if $cl(\lambda)$ is reset in S, then $\chi(\lambda) = true$. From the property of ρ^s that no clock that is reset is checked in a guard, for every transition $r_{m-1} \xrightarrow{cl(\neg\lambda)\geq 1} r_m$ in S, it is clock $cl(\lambda)$ that is reset and hence $\chi(\lambda) = true$. By construction of $\mathcal{A}^Z_\phi$, λ is a literal in C_m. Therefore, we get a literal that is true in every clause evaluating ϕ to true. □

It remains to show that $ZG^{LU}(\mathcal{A}^Z_\phi)$ can also be calculated in polynomial time from $\mathcal{A}^Z_\phi$. We indeed note that the size of the $ZG^{LU}(\mathcal{A}^Z_\phi)$ is the same as that of the automaton. That will conclude the proof that a polynomial algorithm for ZP^{LU} yields a polynomial algorithm for the 3SAT problem.

Theorem 9.2.3 *The zone graph $ZG^{LU}(\mathcal{A}^Z_\phi)$ is isomorphic to $\mathcal{A}^Z_\phi$. The Zenoness problem ZP^{a} is* NP*-complete for $Extra_{LU}$ and $Extra^+_{LU}$.*

Proof
By looking at the guards in the transitions, we get that for each clock x, $L(x) = 1$ and $U_x = -\infty$. The initial node of the zone graph $ZG^{LU}(\mathcal{A}^Z_\phi)$ is $(q_0, Extra_{LU}(Z_0))$ where Z_0 is the set of valuations given by $(x_1 \geq 0) \wedge (x_1 = \overline{x_1} = \dots = x_k = \overline{x_k})$. By definition, since for each clock x, $U_x = -\infty$, we have $Extra_{LU}(Z_0) = \mathbb{R}^X_{\geq 0}$, the non-negative half-space.

On taking a transition with a guard $x \geq 1$ from $\mathbb{R}^X_{\geq 0}$, we come to a zone $\mathbb{R}^X_{\geq 0} \wedge x \geq 1$. However, since $U_x = -\infty$, $Extra_{LU}(\mathbb{R}^X_{\geq 0} \wedge x \geq 1)$ gives back $\mathbb{R}^X_{\geq 0}$. Same for transitions that reset a clock. It follows that $ZG^{LU}(\mathcal{A}^Z_\phi)$ is isomorphic to $\mathcal{A}^Z_\phi$. This extends to $Extra^+_{LU}$ as it is coarser. NP-hardness then comes from Lemma 9.2.2. NP-membership is proved in Lemma 9.2.1. □

9.2.1 Weakening the LU-abstractions

We saw in Lemma 9.2.1 that the extrapolation $Extra_{LU}$ is not lift-safe. This is due to clocks x that are lifted but have $U_x = -\infty$. These are exactly

the clocks x with $L_x \geq 1$ and $U_x = -\infty$. We propose to weaken the U bounds so that the information about a clock being lifted is remembered in the abstracted zone.

Definition 9.2.4 (Weak U bounds) Given the bounds L_x and U_x for each clock $x \in X$, the *weak upper bound* $\overline{U}_x$ is given by: $\overline{U}_x = 1$ if $L_x \geq 1$ and $U_x = -\infty$, and $\overline{U}_x = U_x$ otherwise.

Let $Extra_{L\overline{U}}$ denote the $Extra_{LU}$ abstraction, but with $\overline{U}$ bound for each clock instead of U. This definition ensures that for all lifted clocks, that is, for all $x \in \mathrm{Lf}(\mathcal{A})$, if a zone entails that $x \geq 1$ then $Extra_{L\overline{U}}(Z)$ also entails that $x \geq 1$. This is summarized by the following lemma, the proof of which follows by definitions.

Lemma 9.2.5 For all zones Z, $Extra_{L\overline{U}}$ is lift-safe.

From Theorem 9.1.6, we get that the Zenoness problem is polynomial for $Extra_{L\overline{U}}$. However, there is a price to pay. Weakening the U bounds leads to zone graphs exponentially bigger in some cases. For example, for the automaton $\mathcal{A}^Z_\phi$ that was used to prove the NP-completeness of the Zenoness problem with $Extra_{LU}$, note that the zone graph $ZG^{L\overline{U}}(\mathcal{A}^Z_\phi)$ obtained by applying $Extra_{L\overline{U}}$ is exponentially bigger than $ZG^{LU}(\mathcal{A}^Z_\phi)$. This leads to a slow zone graph $\mathcal{S}ZG^{L\overline{U}}(\mathcal{A}^Z_\phi)$ with size polynomial in $ZG^{L\overline{U}}(\mathcal{A}^Z_\phi)$.

For the abstraction $Extra_{L\overline{U}}$, the abstraction makes use of weak U bounds. Notice that if all clocks that are checked for a lower bound guard are also checked for an upper bound then the two abstractions coincide. So, the wide class of systems where each clock is both bounded from above (i.e. $x \leq c$) and from below (i.e. $x \geq c'$) have polynomial-time detection of Zeno runs, even using $Extra_{LU}$ and $Extra^+_{LU}$.

9.3 Concluding remarks

In this chapter, we have provided the first complete solution to the Zenoness problem. For an automaton $\mathcal{A}$, we construct on-the-fly its slow zone graph $\mathcal{S}ZG^{\mathfrak{a}}(\mathcal{A})$ (Definition 9.1.3). We have shown that the automaton has a Zeno run iff its slow zone graph has cycle of a particular form (Theorem 9.1.6). The slow zone graph solution works with a complexity linear in the size of the abstract zone graph $ZG^{\mathfrak{a}}$ for a class of abstractions called lift-safe abstractions (Definition 9.1.2). We have shown that the abstractions $Extra_M$ and $Extra^+_M$ are lift-safe (Lemma 9.1.7). However, the LU-abstractions are not lift-safe and in particular, we have shown that given the automaton $\mathcal{A}$ and its LU-abstract zone graph, deciding Zenoness is NP-complete (Theorem 9.2.3). We have provided a weakening of the LU-abstractions that is still lift-safe (Definition 9.2.4, Lemma 9.2.5).

In Appendix B, we consider the decision problem:

Given an automaton $\mathcal{A}$, does it have a Zeno run?

Note the difference in the inputs to the problem from the decision problem $\mathsf{ZP}^{\mathfrak{a}}$ defined in page 157. In the above problem, the input is only $\mathcal{A}$ whereas in $\mathsf{ZP}^{\mathfrak{a}}$ the inputs were $\mathcal{A}$ and $ZG^{\mathfrak{a}}(\mathcal{A})$. We show that the above decision problem is PSPACE-complete. This implies that for inferring Zeno runs from an automaton, we need an object as complex as the zone graph.

As in the case of non-Zenoness, it is very striking that the question of Zenoness should depend heavily on abstractions, to the extent of moving from polynomial to NP-complete. In contrast, the fundamental notions of reachability and Büchi emptiness over abstract zone graphs have a mere linear complexity, independent of the abstraction. While working with abstract zone graphs, coarse abstractions (and hence small abstract zone graphs) are essential to handle big models of timed automata. These, as we have seen, work against the Zenoness questions *in the general case*. Our results therefore provide a theoretical motivation to look for cheaper substitutes to the notion of Zenoness.

Chapter 10

Conclusion

We considered the reachability and liveness problems for the timed automata model introduced by [AD94].

The first part of the thesis deals with the reachability algorithm. We have recalled the state-of-the-art algorithm for reachability Section 2.7. The standard algorithm uses the convex $Extra^+_{LU}$ abstraction (Figure 2.12). The LU-bounds are computed for each state q of the automaton by means of a static analysis. This static analysis approach has been discussed in Section 2.6.

We have shown that non-convex abstractions can be used as efficiently as the currently used convex abstractions. To be able to use a non-convex abstraction $\mathfrak{a}$, we need an efficient inclusion test $Z \subseteq \mathfrak{a}(Z')$ (Section 3.1). We have given such efficient inclusion tests for non-convex abstractions $\mathfrak{a}_{\preccurlyeq LU}$ and $Closure_M$ (Theorem 4.5.3, Corollary 4.6.1). If X denotes the set of clocks, the inclusion test involves an $\mathcal{O}(|X|^2)$ number of simple checks as in the case of $Z \subseteq Z'$ and hence can be efficiently implemented in practice. As $\mathfrak{a}_{\preccurlyeq LU}$ and $Closure_M$ are coarser than their convex counterparts, they could potentially yield fewer nodes during the forward exploration.

As a surprising bonus, we have proved that $\mathfrak{a}_{\preccurlyeq LU}$ is optimal: it is the biggest abstraction that is sound and complete for reachability, if the only knowledge about an automaton is the LU-information. This has been the topic of Section 3.2 and the final result appears in Theorem 3.2.10. As a corollary, we obtain that $Closure_M$ is the optimal sound and complete M-abstraction (Theorem 3.4.1).

This optimality result shows that given the LU-bounds or M-bounds information, we cannot do better than $\mathfrak{a}_{\preccurlyeq LU}$ or $Closure_M$ respectively. However, we could still try for better bounds themselves. The smaller the bounds, the bigger the abstractions are. The current method for computing bounds performs a static analysis on the automaton and assigns bound functions to each state q. We propose a new algorithm in Section 5.1 that assigns bound functions on-the-fly during the reachability analysis to every

node (q, Z) of the tree computed. The immediate gain is that constants from unreachable parts of the automaton are taken out of consideration. We have implemented our algorithm in a prototype tool. We have compared results of our algorithm with the standard algorithm in Table 5.1 and have discussed on the gains obtained in Section 5.2.

Relevant publications include [HKSW11] and [HSW12a]. The former contains the constant propagation algorithm and an efficient inclusion test for a non-convex abstraction $Closure^{+}_{LU}$ which is the region closure of the $Extra^{+}_{LU}$ abstraction. The latter paper contains the results with the $\mathfrak{a}_{\preccurlyeq_{LU}}$ abstraction: its optimality and the efficient inclusion test.

The second part of the thesis deals with the liveness questions for timed automata. While considering infinite executions of timed automata, it is important to detect Zeno and non-Zeno runs. We considered two problems in this part: Büchi non-emptiness problem and the problem of deciding existence of Zeno runs.

The Büchi non-emptiness problem asks if there exists a non-Zeno infinite run of the automaton visiting an accepting state infinitely often. The standard solution to this problem involves adding an extra clock to take care of non-Zenoness. This is called the strongly non-Zeno construction. We have recalled this construction in Section 6.4. We have shown that this seemingly harmless construction can lead to an exponential blowup: we have exhibited an example of an automaton $\mathcal{A}_n$ whose simulation graph has size linear in the number of clocks, but the strongly non-Zeno transformation $SNZ(\mathcal{A}_n)$ has a simulation graph whose size is exponential in the number of clocks (Section 7.1).

Subsequent to this observation, we proposed a different solution to non-Zenoness checking in Section 7.2 that makes use of an intermediary guessing zone graph to detect non-Zenoness. For the $Extra_M$ abstraction, the guessing zone graph has size bounded by $|ZG^M(\mathcal{A})|.(|X| + 1)$ where $|ZG^M(\mathcal{A})|$ is the size of the zone graph obtained using $Extra_M$ abstraction (Theorem 7.3.5). We have generalized the guessing zone graph construction and identified a class of abstractions that maintain a polynomial complexity (Theorem 7.3.11). We observed that for LU-extrapolations, the non-Zenoness problem is NP-complete (Theorem 7.4.2). However, a slight weakening of the LU-extrapolations deems the non-Zenoness problem polynomial (Lemma 7.4.5).

Following this complexity analysis, we proceeded to give an on-the-fly algorithm for the Büchi emptiness problem in Chapter 8 using the guessing zone graph construction. We noticed that although the guessing zone graph provides a way to detect non-Zeno runs, it is useful only when the automaton indeed contains zero-checks. We then provided an optimized algorithm to use the guessing zone graph construction only when necessary in Section 8.2. Experimental results showing the gain of our algorithm over the standard

strongly non-Zeno constructions have been reported in Section 8.3.

In Chapter 9, we considered the problem of deciding existence of Zeno runs given a timed automaton. The current solution to this problem involves a static check on the automaton to find loops satisfying certain conditions. If every loop of the automaton satisfies this condition, then there are no Zeno runs in the automaton. However, there could be automata with loops not satisfying this condition, but still there are no Zeno runs in the automata. We have recalled this criterion in detail in Section 6.5. We have proposed a complete solution to this problem by lifting this syntactic criterion to the zone graph. The solution we propose is linear in the size of the zone graph for a class of abstractions called lift-safe (Theorem 9.1.6). The M-extrapolations and a slight weakening of LU-extrapolations fall in this class (Lemmas 9.1.7 and 9.2.5). For LU-extrapolations, the Zenoness problem turns out to be NP-complete (Theorem 9.2.3).

Relevant publications include [HSW12b] and [HS11]. The former contains the guessing zone graph construction and the on-the-fly algorithm. It considers only M-abstractions. It is a combined long version of [HSW10] and [HS10]. The latter work [HS11] deals with non-Zenoness problem on bigger abstractions. The NP-completeness of LU-abstractions appears here. The results on the Zenoness problem appear in this work too.

Perspectives

The results developed in the thesis open roads to numerous research directions. We list a selected few below.

More than LU

The optimality of $\mathfrak{a}_{\preccurlyeq LU}$ with respect to LU-bounds may seem to suggest that the topic is closed. Indeed, it shows that one cannot do better from the abstractions side when the only available knowledge includes just the LU-bounds. We have circumvented this situation to some extent by giving a way to obtain better LU-bounds by propagating constants on-the-fly during exploration. This constant propagation algorithm has a very convenient property that it maintains the original unabstracted zones at each node. The structure of this algorithm gives the liberty to abstract the zone on-the-fly using whatever information one can garner during the reachability analysis. The current algorithm collects constants occurring in guards. Let us say that this algorithm assigns L_x to be 10 at some node. Then this means that in the subtree below the node, all guards $x \geqslant 1, x \geqslant 2, \ldots, x \geqslant 10$ are potentially present and additionally paths using different orders of these guards are possible. The simulation relation $\preccurlyeq_{LU}$ on which $\mathfrak{a}_{\preccurlyeq LU}$ is based on is defined assuming all these possibilities. However, in the course of the reachability analysis, we know a lot more than this. We know the exact

subtree present below the node; the exact guards that are present and the exact order in which they occur. An interesting line of research could be to come up with efficient abstractions using more of the semantic information than just LU-bounds.

Diagonal constraints

Automata with diagonal constraints have till date been very challenging. These are timed automata in which guards of the from $x - y \sim c$ are allowed. It is known that diagonal constraints do not add expressive power to timed automata and can be removed [AD94, BDPG98]. However, the resulting automaton blows up the original automaton by a size exponential in the number of diagonal constraints. This construction is not feasible in practice.

The standard forward analysis algorithm using $Extra_M$ had been in use even for automata with diagonal constraints in the late 90s [DT98]. But this abstraction was shown not to be sound for diagonal constraints by [Bou03, Bou04]. A way to split zones abstracted with $Extra_M$ by diagonal constraints occurring in the automaton was proposed in [BY04]. In the worst case the complexity of this algorithm is the same as that of the region graph construction. An abstraction-refinement based method has been proposed by [BLR05], which is the most recent work on diagonal constraints, to our knowledge.

In [Bou04], a new region construction is given that is sound and complete for automata with diagonal constraints. Call it d-regions. A $Closure^d$ abstraction which for a zone takes the union of all these d-regions intersecting it, is shown to be sound and complete. This $Closure^d$ abstraction is known to be non-convex as well. An immediate question would be to try an efficient inclusion test for the $Closure^d$ abstraction. Looking at LU-abstractions for automata with diagonal constraints could follow.

Extensions of timed automata

Timed automata are the basis of many different models: updatable timed automata [BDFP04, Bou04]; probabilistic timed automata [KNP11, CHK08, Bou09] and priced timed automata [BFLM11]. Model-checking algorithms for these extensions go via a zone based approach. This immediately implies that abstractions are of prime importance in these models too. To what extent the results of this thesis apply to these extended models, is another non-trivial task on hand.

Efficient liveness

In the liveness part of the thesis, we concentrated solely on the questions related to Zeno and non-Zeno executions. In particular, we restricted our focus towards convex abstractions as is the case with standard literature

on the problem. Theorem 6.3.3 proposed by [Li09] shows that repeated state reachability can be decided by using any abstraction based on a time-abstract simulation. This is a very generic theorem and in particular, it allows the use of non-convex $\mathfrak{a}_{\preccurlyeq LU}$ and $Closure_M$ abstractions. However, the standard algorithm requires to represent these abstractions and hence would not be efficient. We need our proposed schema for non-convex abstractions as in Section 3.1 to efficiently use these abstractions: that is, store the unabstracted zones and use non-convex abstractions indirectly through inclusion tests. The generic theorem along with the transition compatibility condition (c.f. Section 3.1) would ensure correctness of this approach for liveness checking. However, we don't yet have an implementation of non-convex abstractions for liveness, validating the gains of using them. Additionally, the other optimization in the reachability part involving on-the-fly calculation of bounds needs to be investigated for liveness checking. We plan to combine these two optimizations: non-convex abstractions and on-the-fly bounds for liveness, in a future implementation.

Timed games

There is a lot of literature on timed games [MPS95, DAFH$^+$03, CHP10, CDF$^+$05]. A tool UPPAAL-TIGA [BCD$^+$07] has been implemented to solve games with respect to reachability and safety properties. A direction for future work would be to consider the question of a zone-based approach for liveness properties in timed games and in particular, synthesis of non-Zeno strategies using the constructions developed in this thesis. The liveness question has been considered in [DAFH$^+$03] and [CHP10], but the algorithms are not zone based. Additionally, the question of non-Zenoness is solved by adding an extra clock, that as we show here, may induce an exponential blowup.

This brings us to the final concluding remarks. In this thesis, we have seen how to efficiently handle abstractions for timed automata, and to what extent can one push abstractions to facilitate correct verification of some of the fundamental properties. We have proposed modifications to the standard reachability and liveness algorithms and have noticed substantial gains in our approaches. A common feature in the observed gains for reachability and liveness is the role of "semantic information" in pruning the search. A lot more of it remains to be exploited.

References

[ACH+92] R. Alur, C. Courcoubetis, N. Halbwachs, D. Dill, and H. Wong-Toi. Minimization of timed transition systems. In *CONCUR'92*, pages 340–354. Springer, 1992.

[AD94] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[AM04] R. Alur and P. Madhusudan. Decision problems for timed automata: A survey. In M. Bernardo and F. Corradini, editors, *Formal Methods for the Design of Real-Time Systems, International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM-RT 2004, Bertinoro, Italy, September 13-18, 2004, Revised Lectures*, volume 3185 of *Lecture Notes in Computer Science*, pages 1–24. Springer, 2004.

[BBFL03] G. Behrmann, P. Bouyer, E. Fleury, and K. G. Larsen. Static guard analysis in timed automata verification. In *TACAS'03*, volume 2619 of *LNCS*, pages 254–270. Springer, 2003.

[BBLP06] G. Behrmann, P. Bouyer, K. G. Larsen, and R. Pelanek. Lower and upper bounds in zone-based abstractions of timed automata. *Int. Journal on Software Tools for Technology Transfer*, 8(3):204–215, 2006.

[BBP04] B. Bérard, B. Bouyer, and A. Petit. Analysing the pgm protocol with UPPAAL. *Int. Journal of Production Research*, 42(14):2773–2791, 2004.

[BCD+07] G. Behrmann, A. Cougnard, A. David, E. Fleury, K.G. Larsen, and D. Lime. Uppaal-tiga: Time for playing games! In *Proceedings of the 19th international conference on Computer aided verification*, pages 121–125. Springer-Verlag, 2007.

[BDFP04] P. Bouyer, C. Dufourd, E. Fleury, and A. Petit. Updatable timed automata. *Theoretical Computer Science*, 321(2):291–345, 2004.

[BDL+06] G. Behrmann, A. David, K. G Larsen, J. Haakansson, P. Pettersson, W. Yi, and M. Hendriks. Uppaal 4.0. In *QEST'06*, pages 125–126, 2006.

[BDM+98] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. KRONOS: a mode-checking tool for real-time systems. In *CAV'98*, volume 1427 of *LNCS*, pages 546–550. Springer, 1998.

[BDPG98] B. Bérard, V. Diekert, A. Petit, and P. Gastin. Characterization of the expressive power of silent transitions in timed automata. *Fundamenta Informaticae*, 36(2):145–182, 1998.

[BFLM11] P. Bouyer, U. Fahrenberg, K.G.. Larsen, and N. Markey. Quantitative analysis of real-time systems using priced timed automata. *Communications of the ACM*, 54:78–87, 2011.

[BG06] H. Bowman and R. Gómez. How to stop time stopping. *Formal aspects of computing*, 18(4):459–493, 2006.

[BLR05] P. Bouyer, F. Laroussinie, and P.A. Reynier. Diagonal constraints in timed automata: Forward analysis of timed systems. *Formal Modeling and Analysis of Timed Systems*, pages 112–126, 2005.

[Bou03] P. Bouyer. Untameable timed automata! *STACS 2003*, pages 620–631, 2003.

[Bou04] P. Bouyer. Forward analysis of updatable timed automata. *Form. Methods in Syst. Des.*, 24(3):281–320, 2004.

[Bou09] P. Bouyer. *From Qualitative to Quantitative Analysis of Timed Systems.* Mémoire d'habilitation, Université Paris 7, Paris, France, 2009.

[BY04] J. Bengtsson and W. Yi. Timed automata: Semantics, algorithms and tools. *Lectures on Concurrency and Petri Nets*, pages 87–124, 2004.

[CDF+05] Franck Cassez, Alexandre David, Emmanuel Fleury, Kim Guldstrand Larsen, and Didier Lime. Efficient on-the-fly algorithms for the analysis of timed games. In *CONCUR*, pages 66–80, 2005.

[CDLP05] J.-M. Couvreur, A. Duret-Lutz, and D. Poitrenaud. On-the-fly emptiness checks for generalized Büchi automata. In

P. Godefroid, editor, *Model Checking Software, 12th International SPIN Workshop, San Francisco, CA, USA, August 22-24, 2005, Proceedings*, volume 3639 of *Lecture Notes in Computer Science*, pages 169–184. Springer, 2005.

[CHK08] T. Chen, T. Han, and J.-P. Katoen. Time-abstracting bisimulation for probabilistic timed automata. In *TASE*, pages 177–184. IEEE Computer Society, 2008.

[CHP10] Krishnendu Chatterjee, Thomas A. Henzinger, and Vinayak S. Prabhu. Timed parity games: Complexity and robustness. *Logical Methods in Computer Science*, 7(4), 2010.

[Cou99] J.-M. Couvreur. On-the-fly verification of linear temporal logic. *FM'99 - Formal Methods, World Congress on Formal Methods in the Development of Computing Systems, Toulouse, France, September 20-24, 1999, Proceedings, Volume I*, 1708:253–271, 1999.

[CY92] C. Courcoubetis and M. Yannakakis. Minimum and maximum delay problems in real-time systems. *Form. Methods Syst. Des.*, 1(4):385–415, 1992.

[DAFH+03] L. De Alfaro, M. Faella, T. Henzinger, R. Majumdar, and M. Stoelinga. The element of surprise in timed games. *CONCUR 2003-Concurrency Theory*, pages 144–158, 2003.

[Dil89] D. Dill. Timing assumptions and verification of finite-state concurrent systems. In *AVMFSS*, volume 407 of *LNCS*, pages 197–212. Springer, 1989.

[DT98] C. Daws and S. Tripakis. Model checking of real-time reachability properties using abstractions. In *TACAS'98*, volume 1384 of *LNCS*, pages 313–329. Springer, 1998.

[GB07] R. Gómez and H. Bowman. Efficient detection of zeno runs in timed automata. In *FORMATS*, volume 4763 of *LNCS*, pages 195–210, 2007.

[GJ95] H. Gregersen and H.E. Jensen. *Formal design of reliable real time systems*. Master's thesis, Department of Mathematics and computer Science, Aalborg University, 1995.

[Gom06] Rodolfo Sabas Gomez. *Verification of Real-Time Systems: Improving Tool Support*. PhD thesis, Computing Laboratory, University of Kent, October 2006.

[GS09] A. Gaiser and S. Schwoon. Comparison of algorithms for checking emptiness on Büchi automata. In P. Hilený, V. Matyás, and T. Vojnar, editors, *Annual Doctoral Workshop on Mathematical and Engineering Methods in Computer Science, MEMICS 2009, November 13-15, 2009, Prestige Hotel, Znojmo, Czech Republic*, volume 13 of *OASICS*, pages 69–77. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2009.

[HBL+04] M. Hendriks, G. Behrmann, K. G. Larsen, P. Niebert, and F. Vaandrager. Adding symmetry reduction to Uppaal. In *Int. Workshop on Formal Modeling and Analysis of Timed Systems*, volume 2791 of *LNCS*, pages 46–59. Springer, 2004.

[HKSW11] F. Herbreteau, D. Kini, B. Srivathsan, and I. Walukiewicz. Using non-convex approximations for efficient analysis of timed automata. In *FSTTCS*, volume 13 of *LIPIcs*, pages 78–89. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

[HNSY94] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. *Inf. Comput.*, 111(2):193–244, 1994.

[HS10] F. Herbreteau and B. Srivathsan. Efficient On-the-Fly Emptiness Check for Timed Büchi Automata. In A. Bouajjani and W.-N. Chin, editors, *Automated Technology for Verification and Analysis: 8th International Symposium, ATVA 2010, Singapore, September 21-24, 2010, Proceedings*, volume 6252 of *Lecture Notes in Computer Science*, pages 218–232. Springer, 2010.

[HS11] Frédéric Herbreteau and B. Srivathsan. Coarse abstractions make zeno behaviours difficult to detect. In *CONCUR*, pages 92–107, 2011.

[HSLL97] K. Havelund, A. Skou, K. Larsen, and K. Lund. Formal modeling and analysis of an audio/video protocol: An industrial case study using UPPAAL. In *RTSS'97*, pages 2–13, 1997.

[HSW10] Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. Efficient emptiness check for timed büchi automata. In *CAV*, pages 148–161, 2010.

[HSW12a] Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. Better abstractions for timed automata. In *LICS*, 2012.

[HSW12b] Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. Efficient emptiness check for timed büchi automata. *Formal Methods in System Design*, 40(2):122–146, 2012.

[HU79] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[KNP11] M. Kwiatkowska, G. Norman, and D. Parker. Prism 4.0: Verification of probabilistic real-time systems. In *CAV*, volume 6806 of *LNCS*, pages 585–591, 2011.

[KNSS02] M. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic verification of real-time systems with discrete probability distributions. *Theoretical Computer Science*, 282(1):101–150, 2002.

[Li09] Guangyuan Li. Checking timed büchi automata emptiness using lu-abstractions. In Joël Ouaknine, editor, *Formal modeling and analysis of timed systems. 7th Int. Conf. (FORMATS)*, volume 5813 of *Lecture Notes in Computer Science*, pages 228–242. Springer, 2009.

[LS00] F. Laroussinie and Ph. Schnoebelen. The state-explosion problem from trace to bisimulation equivalence. In *FoSSaCS*, volume 1784 of *LNCS*, pages 192–207. Springer, 2000.

[MPS95] O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems. In *STACS 95*, pages 229–242. Springer, 1995.

[MPS11] G. Morbé, F. Pigorsch, and C. Scholl. Fully symbolic model checking for timed automata. In *CAV*, volume 6806 of *LNCS*, pages 616–632. Springer, 2011.

[SE05] S. Schwoon and J. Esparza. A note on on-the-fly verification algorithms. In N. Halbwachs and L. D. Zuck, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 11th International Conference, TACAS 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*, volume 3440 of *Lecture Notes in Computer Science*, pages 174–190, 2005.

[TAKB96] S. Tasiran, R. Alur, R. P. Kurshan, and R. K. Brayton. Verifying abstractions of timed systems. In *CONCUR*, volume 1119 of *LNCS*, pages 546–562. Springer, 1996.

[Tri99] S. Tripakis. Verifying progress in timed systems. In J.-P. Katoen, editor, *Formal Methods for Real-Time and Probabilistic Systems, 5th International AMAST Workshop, ARTS'99, Bamberg, Germany, May 26-28, 1999. Proceedings*, volume

1601 of *Lecture Notes in Computer Science*, pages 299–314. Springer, 1999.

[TY01] S. Tripakis and S. Yovine. Analysis of timed systems using time-abstracting bisimulations. *Form. Methods Syst. Des.*, 18:25–68, 2001.

[TYB05] S. Tripakis, S. Yovine, and A. Bouajjani. Checking timed Büchi automata emptiness efficiently. *Formal Methods in System Design*, 26(3):267–292, 2005.

[Wan04] Farn Wang. Efficient verification of timed automata with BDD-like data structures. *Int. J. on Software Tools for Technology Transfer*, 6:77–97, 2004.

[YL93] M. Yannakakis and D. Lee. An efficient algorithm for minimizing real-time transition systems. In *Computer Aided Verification*, pages 210–224. Springer, 1993.

[ZLZ05] J. Zhao, X. Li, and G. Zheng. A quadratic-time dbm-based successor algorithm for checking timed automata. *Inf. Process. Lett.*, 96(3):101–105, 2005.

Appendix A

Choice of semantics

The choice of semantics for the symbolic transition of Definition 2.3.1 differs from standard semantics in the literature [DT98, Bou09, TYB05] in the order of time and action transitions. We would like to briefly explain our choice of semantics.

Let $\overrightarrow{W}$ denote the set of all time-successors of W:

$$\overrightarrow{W} = \{v + \delta \mid v \in W, \delta \in \mathbb{R}_{\geq 0}\}$$

We say that a set W is time-elapsed if $W = \overrightarrow{W}$.

Let us denote the symbolic transition relation (Post_t) of [Bou09] as $\Rightarrow_s^t$. For every transition t and every set of valuations W, there is a transition $(q, W) \Rightarrow_s^t (q', W')$ where:

$$W' = \{v' \mid \exists v \in W,\ \exists \delta \in \mathbb{R}_{\geq 0}.\ (q, v) \rightarrow^{\delta} \rightarrow^{t} (q', v')\}$$

The only difference is that in $\Rightarrow_s^t$ we consider the set of valuations v' such that there is a valuation $v \in W$ that can elapse some time δ and take transition t, that is: $(q, v) \rightarrow^{\delta} \rightarrow^{t} (q', v')$. The set W' therefore is not necessarily time-elapsed.

In the symbolic transition $\Rightarrow^t$ that we defined above, we chose to have the set of valuations v' such that there is a valuation $v \in W$ from which we can directly take the transition t and then elapse some time δ, that is: $(q, v) \rightarrow^{t} \rightarrow^{\delta} (q', v')$. The set W' obtained here is therefore time-elapsed.

Consider the following path given by the relation $\Rightarrow_s^t$:

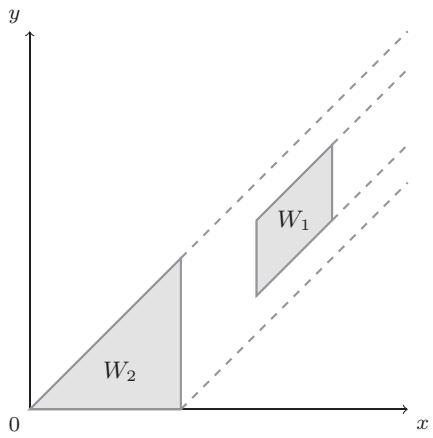
$$(q_0, W_0) \Rightarrow_s^{t_1} (q_1, W_1) \Rightarrow_s^{t_2} \ldots \Rightarrow_s^{t_n} (q_n, W_n)$$

In our case, we start with a time-elapsed set $\overrightarrow{W_0}$ and we get the corresponding time-elapsed sets when the same sequence of transitions is considered.

$$(q_0, \overrightarrow{W_0}) \Rightarrow^{t_1} (q_1, \overrightarrow{W_1}) \Rightarrow^{t_2} \ldots \Rightarrow^{t_n} (q_n, \overrightarrow{W}_n)$$

Consider the forward exploration algorithm to compute the transition system. When a node (q_i, W_i) is reached for which there is an already computed node (q_j, W_j) such that $q_i = q_j$ and $W_i \subseteq W_j$, this node is not considered for further computation as both $\Rightarrow_s^t$ and $\Rightarrow^t$ are monotone with respect to inclusion.

Observe that if $W_i \subseteq W_j$ then $\overrightarrow{W_i} \subseteq \overrightarrow{W_j}$ but the converse is not true. We illustrate an example in the figure shown below.



The figure shows two sets W_1, W_2. The dashed lines show the boundaries of $\overrightarrow{W_1}$ and $\overrightarrow{W_2}$. Observe that $\overrightarrow{W_1} \subseteq \overrightarrow{W_2}$ but $W_1 \not\subseteq W_2$. Therefore we hope to get faster inclusions by maintaining time-elapsed sets. This motivates our definition of the symbolic transition (Definition 2.3.1).

Another common notion of semantics has been to differentiate between time-elapse transitions and action transitions [BBLP06, BBFL03]. Yet again, in this semantics, it is sufficient for the reachability algorithm to store only time-elapsed zones and in fact by the previous discussion, is potentially more productive.

There is another important reason for keeping time-elapsed zones. As we see in Theorem 3.3.3, the $\mathfrak{a}_{\preccurlyeq LU}$ abstraction of [BBLP06] is the optimal abstraction when time-elapsed zones are considered.

Appendix B

PSPACE-completeness of (non-)Zeno run detection

We prove the following theorem.

Theorem B.0.1 *Given an automaton $\mathcal{A}$, deciding if there exists a non-Zeno run is* PSPACE*-complete. Similarly for deciding if there exists a Zeno run.*

Our proof follows the same lines as the proof of PSPACE-completeness of the emptiness problem for timed automata [AD94, CY92].

PSPACE-membership

In Lemmas 7.4.3 and 9.2.1 we have proved that given $\mathcal{A}$ and $ZG^{LU}(\mathcal{A})$, there is a non-deterministic polynomial algorithm for NZP^{LU} and ZP^{LU}. Essentially both the algorithms do the following. They begin by non-deterministically guessing a node (q, Z) of $ZG^{LU}(\mathcal{A})$ and augmenting it with a guessed subset of clocks $S \subseteq X$ to give the node (q, Z, S). Starting from this node, the algorithms construct a cycle of $ZG^{LU}(\mathcal{A})$ containing (q, Z) and satisfying certain constraints specified by this newly augmented component:

$$(q, Z, S) \Rightarrow^{t_1} (q_1, Z_1, S_1) \Rightarrow^{t_2} \ldots \Rightarrow^{t_n} (q_n, Z_n, S_n) \Rightarrow^{t} (q, Z, S')$$

Since Z can be represented in space $\mathcal{O}(|X|^2)$ using a DBM, nodes of the form (q, Z, S) can be represented in space polynomial in the size of $\mathcal{A}$. To find the above cycle, it is enough to maintain the initially guessed node (q, Z, S) and the current node whose successor has to be computed. Clearly, the non-deterministic algorithm needs space that is polynomial in the size of the input $\mathcal{A}$. By Savitch's theorem, this shows that deciding if a timed automaton has a non-Zeno run, or dually a Zeno run, is in PSPACE.

PSPACE-hardness

The problem of deciding if a deterministic Linear Bounded Automaton[1] (LBA) $\mathcal{B}$ accepts a word w is known to be PSPACE-complete [HU79]. We reduce the acceptance problem for deterministic LBAs to the problem of deciding if a timed automaton has a Zeno or a non-Zeno run.

Let $\mathcal{B}$ be a deterministic LBA and let w be a finite word on the input alphabet of $\mathcal{B}$. Without loss of generality, we can assume that $\mathcal{B}$ has a single accepting state q_F from which there are no outgoing transitions. We also assume that the tape alphabet of $\mathcal{B}$ is $\Gamma = \{1, \ldots, k-1\}$. Let n be the length of the input word w (hence the size of the tape of $\mathcal{B}$).

We build a timed automaton $\mathcal{A}$ that reads the sequence σ of configurations of $\mathcal{B}$ on input w encoded as:

$$\gamma_1^0\gamma_2^0\cdots\gamma_n^0 \;\; k \;\; \gamma_1^1\gamma_2^1\cdots\gamma_n^1 \;\; k \cdots k \;\; \gamma_1^i\gamma_2^i\cdots\gamma_n^i \;\; k \cdots$$

where:

- $\gamma_1^0\gamma_2^0\cdots\gamma_n^0$ is the word w, which is the initial content in the tape;
- $\gamma_1^i\gamma_2^i\cdots\gamma_n^i$ is the content in the tape after the first i transitions of $\mathcal{B}$.

For every word w, there is a unique encoding σ as $\mathcal{B}$ is deterministic. Observe that σ is a sequence of integers in $1, \ldots, k$ and k acts as a separator between successive configurations. The automaton $\mathcal{A}$ that we construct below accepts the sequence σ iff $\mathcal{B}$ accepts w.

Call $\gamma_1^i\gamma_2^i\cdots\gamma_n^i$ as the i^{th} block. Each block i can be mapped to a position $p_i \in \{1, \ldots, n\}$ which represents the position of the tape head after the i^{th} transition. The position p_0 is the initial position of the tape head which is 1. Similarly, each block i can be mapped to a state q_i of the the LBA $\mathcal{B}$ representing the state of $\mathcal{B}$ after the first i transitions.

We construct the automaton $\mathcal{A}$ as follows.

States.

The states of the automaton encode the state of $\mathcal{B}$ and the position of the tape head. So each state of the automaton is of the form (q, p) where q is a state of $\mathcal{B}$ and $p \in \{1, \ldots, n\}$ is a position of the tape head. There is an extra auxiliary state $(q_{init}, 0)$ to read the initial block of σ which is the word w itself. The goal is to make the automaton come to (q_i, p_i) after reading the first i blocks:

[1] Linear Bounded Automata are Turing Machines with tape bounded by the length of the input word.

$$\underbrace{\gamma_1^0\gamma_2^0\cdots\gamma_n^0\ k}_{(q_{init},0)}\ \underbrace{\gamma_1^1\gamma_2^1\cdots\gamma_n^1\ k}_{(q_0,p_0)}\cdots k\ \underbrace{\gamma_1^i\gamma_2^i\cdots\gamma_n^i\ k}_{(q_{i-1},p_{i-1})}\ \underbrace{\gamma_1^{i+1}\gamma_2^{i+1}\cdots\gamma_n^{i+1}\ k}_{(q_i,p_i)}\cdots$$

The initial block is read in the initial state $(q_{init}, 0)$ after which the automaton moves to (q_0, p_0). In general, after reading block i, the automaton should move to (q_i, p_i) which represents the state q_i of $\mathcal{B}$ and the position p_i of the tape head at the time of taking the i^{th} transition. While reading the $i + 1^{th}$ block from state (q_i, p_i) the automaton has to check if the symbol at position p_i of the block corresponds to the modification of the $i + 1^{th}$ transition of $\mathcal{B}$ which is of the form $(q_i, \gamma, \gamma', \Delta, q_{i+1})$.

Clocks.

We intend to make the automaton $\mathcal{A}$ spend $k+1$ time units at each symbol. This is facilitated by a clock x. Spending $k + 1$ time units will also help us to recognize the current symbol which is a number between 1 and k. To this regard, to read a symbol $s \in \sigma$, we use a transition with guard $(x = s)$ followed by a transition with guard $(x = k + 1)$ that resets x. As reading a symbol requires $(k + 1)$ time units, reading a tape configuration (followed by separator symbol k) takes $(n + 1).(k + 1)$ time units.

To store the currently read symbol, we introduce a clock x_j for each cell j of the tape. If the currently read symbol is γ_j^i, then clock x_j is reset on the transition with guard $x = \gamma_j^i$. Hence, when the symbol γ_j^{i+1} is read, the previous content of the cell j, given by the symbol γ_j^i, is remembered in x_j by the value $(n + 1).(k + 1) - \gamma_j^i + \gamma_j^{i+1}$. This is illustrated in (B.1).

$$\cdots \underbrace{\overbrace{\xrightarrow{(x=\gamma_j^i),\{x_j\}}}^{\gamma_j^i\ t.u.}\ \xrightarrow{(x=k+1),\{x\}}\ \cdots}_{(n+1)\cdot(k+1)\ time\ units}\ \overbrace{\xrightarrow{(x=\gamma_j^{i+1}),\{x_j\}}}^{\gamma_j^{i+1}\ t.u.}\ \xrightarrow{(x=k+1),\{x\}}\ \cdots \tag{B.1}$$

Transitions.

Consider a state (q, p) of $\mathcal{A}$. For each transition $(q, \gamma, \gamma', \Delta, q')$ of $\mathcal{B}$, there is a sequence of transitions in $\mathcal{A}$ that reads a block and does the following:

- ensures that the p^{th} symbol corresponds to the modification of the p^{th} tape cell forced by this transition,
- ensures that all other symbols are left unchanged corresponding to all other cells being unchanged,
- moves to state $(q', p+\Delta)$ after reading the block, if $p+\Delta \in \{1, \ldots, n\}$.

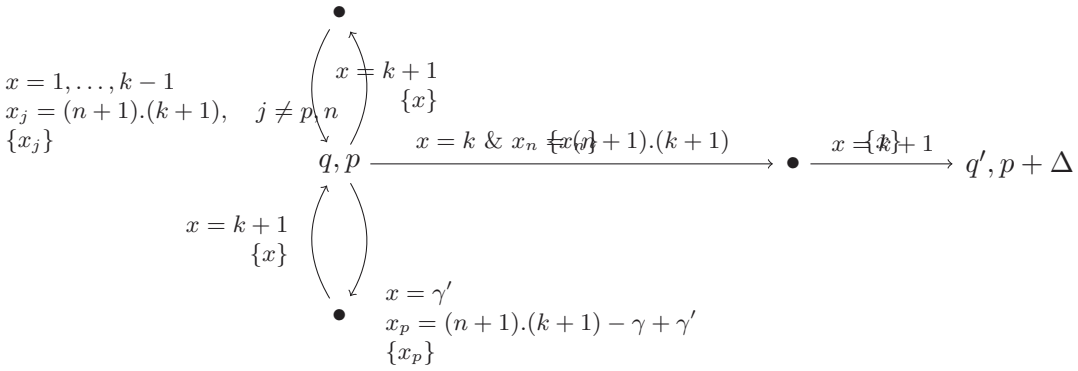


Figure B.1: Widget for transition $(q, \gamma, \gamma', \Delta, q')$ on state (q, p).

Moreover, the cells have to be read in the right order, that is, cell 1 should be read followed by cell 2, etc. Recall that x_j is the clock associated with cell j. For every $j \neq p$, we check if $x_j = (n+1).(k+1)$ and for $j = p$ we check if $x_j = (n+1).(k+1) - \gamma + \gamma'$. This will ensure the first two conditions above and will also ensure that the cells are read in the correct succession.

The complete widget for transition $(q, \gamma, \gamma', \Delta, q')$ is depicted in Figure B.1. There is one such widget in $\mathcal{A}$ for each state (q, p) such that $p + \Delta$ is a valid position (i.e. $p + \Delta \in \{1, \ldots, n\}$).

Initialization

We need to read the word w from state $(q_{init}, 0)$ and assign the initial value of the clocks $x_1, \ldots, x_n$ to $w_1, \ldots, w_n$ where w_j represents the j^{th} symbol of w. As w is given as an input, we can easily add transitions from $(q_{init}, 0)$ to ensure this and jump to $(q_0, 1)$.

Observe that since $\mathcal{B}$ is deterministic, $\mathcal{A}$ is also deterministic. Furthermore, $\mathcal{A}$ is time-deterministic as all the guards are equalities. Hence, $\mathcal{A}$ has a single run given the word w. Furthermore, if $\mathcal{B}$ does not terminate on w, the corresponding run of $\mathcal{A}$ is infinite and non-Zeno.

Recall that q_F is the sole accepting state of $\mathcal{B}$ and there are no transitions outgoing from q_F. From the construction described above, one easily gets the following theorem:

Theorem B.0.2 *$\mathcal{A}$ reaches a state (q_F, p) iff $\mathcal{B}$ reaches q_F on input w. The size of $\mathcal{A}$ is polynomial in the size of $\mathcal{B}$ and w.*

Existence of a Non-Zeno Run

We show that an algorithm for deciding if $\mathcal{A}$ has a non-Zeno run yields an algorithm to decide if $\mathcal{B}$ accepts w. This algorithms has two phases.

In the first phase, it determines if $\mathcal{A}$ has a non-Zeno run:

- if the answer is *yes*, we can conclude that $\mathcal{B}$ does not accept w. Indeed, if $\mathcal{A}$ has a non-Zeno run, then it does not reach (q_F, p) for any p as the run is infinite, hence $\mathcal{B}$ neither reaches q_F;
- if the answer is *no*, we cannot conclude. We only gain information that $\mathcal{A}$ has no infinite run, but it may stop in a state (q_F, p) as well as in a non-accepting state.

In the second phase, we transform $\mathcal{A}$ into $\mathcal{A}'$ by adding a loop on all (q_F, p) with guard $(x \geq 1)$ and that resets x. Now, if the run of $\mathcal{A}'$ is infinite, then it visits some (q_F, p). Furthermore, it is the only non-Zeno run in $\mathcal{A}'$ as we know from the first phase that $\mathcal{A}$ has no infinite run. We now ask if $\mathcal{A}'$ has a non-Zeno run:

- if the answer is *yes*, we can conclude that $\mathcal{B}$ accepts w;
- if the answer is *no*, the run of $\mathcal{A}'$ is finite and does not reach any (q_F, p). We can conclude that $\mathcal{B}$ does not accept w.

Existence of a Zeno Run

Now, we show that an algorithm that decides if $\mathcal{A}$ has a Zeno run yields an algorithm to decide if $\mathcal{B}$ accepts w. Recall that $\mathcal{A}$ is deterministic: it has a unique run and if that run is infinite, then it is non-Zeno.

We transform $\mathcal{A}$ into $\mathcal{A}'$ by adding a loop on all states (q_F, p) with guard $(x \leq 0)$. Then we asks if $\mathcal{A}'$ has a Zeno run:

- if the answer is *yes*, then some (q_F, p) has to be reachable, hence $\mathcal{B}$ reaches q_F and accepts w;
- if the answer is *no*, then no (q_F, p) is reachable, and $\mathcal{B}$ does not accept w.

Appendix C

Couvreur's algorithm

The Couvreur's algorithm shown in Figure C.1, can be viewed as three functions *check_scc*, *merge_scc* and *close_scc*. The algorithm annotates every node with an integer $dfsnum$ and a boolean *opened*. The variable $dfsnum$ is assigned based on the order of appearance of the nodes during the depth-first search. The *opened* bit is set to *true* when the node is just opened for exploration by *check_scc* and is set to false by *close_scc* when the maximal SCC of the node has been completely explored. The algorithm uses two stacks *Roots* and *Active*. The *Roots* stack stores the root of each SCC in the current search path. The root is the node of the SCC that was first visited by the DFS. If the roots stack is $s_0 s_1 \dots s_n$, then for $0 \leq i \leq n-1$, s_i is the root of the SCC containing all the nodes with $dfsnum$ between $s_i.dfsnum$ and $s_{i+1}.dfsnum$ that have *opened* set to *true* and s_n is the root of the SCC containing all nodes with $dfsnum$ greater than $s_n.dfsnum$ which have *opened* set to *true*.

The main function *check_scc* proceeds by exploring the graph in depth-first search (DFS) order. When a successor t of the current node s is found, $t.dfsnum$ being zero implies t has not been visited yet and $t.opened$ being true implies that t belongs to the same SCC as s. When t belongs to the SCC of s, all the nodes visited in the path from t to s also belong to the SCC of s. These nodes are collected by the function *merge_scc*, which finds the root s_i of t and repeatedly pops the *Roots* stack so that s_i comes to the top, signifying that it is the root of the SCC containing all the nodes visited from t to s. A *maximal* SCC is detected when all the transitions of the current node s have been explored, with s being on the top of the *Roots* stack. The *close_scc* function is now called that sets the *opened* bit of all nodes in the SCC rooted at s to $false$. To identify these nodes, the *Active* stack is used, which stores all the nodes of the partially explored SCCs, in the order of the $dfsnum$.

```
1  function emptiness_check()
2        count := 0; Roots, Active := ∅
3        check_scc(s_0)
4        report L(A) = ∅
5
6  function check_scc(s)
7        count++; s.dfsnum := count
8        s.opened := ⊤;
9        Add (s, s.labels) to Roots
10       Add s to Active
11       for all s → t do
12             if (dfsnum = 0)
13                   check_scc(t)
14             else if (t.opened)
15                   merge_scc(t)
16       if top(Roots) = (s,···)
17             close_scc()
18
19 function merge_scc(t)
20       A:=∅;
21       repeat
22             Remove (s,a) from (Roots)
23             A:=A∪a
24       until s.dfsnum ≤ t.dfsnum
25       Add (s, A) to Roots
26       if (Acc⊆A)
27             report L(A) ≠ ∅
28
29 function close_scc()
30        Remove (s, a) from Roots
31        repeat
32             Remove u from Active
33             u.opened := ⊥
34        until u = s
```

Figure C.1: The Couvreur's Algorithm for Emptiness Check of Büchi Automata. States of the automaton are denoted s,t,... The transition relation is $\rightarrow$. The set of accepting states is Acc. The initial state is s_0. The accepting label on state s is given by s.labels