THESIS IN COTUTELLE PRESENTED

TO OBTAIN THE DEGREE OF

Doctor of Philosophy to

UNIVERSITY OF BORDEAUX

and CHENNAI MATHEMATICAL INSTITUTE

DOCTORAL SCHOOL OF MATHEMATICS AND COMPUTER SCIENCE

By

Govind RAJANBABU

PARTIAL ORDER REDUCTION FOR TIMED SYSTEMS

Under the supervision of **Frédéric HERBRETEAU**

and **B SRIVATHSAN**

and **Igor WALUKIEWICZ**

Defended on 16 June 2021

Members of jury :

Thomas CHATAIN	Maître de Conférences	ENS Paris-Saclay	Reviewer
Silvano DAL ZILIO	Chargé de Recherche,	LAAS-CNRS Toulouse	Examiner
Frédéric HERBRETEAU	Maître de Conférences	Université de Bordeaux	Codirector
David ILCINKAS	Chargé de Recherche	Université de Bordeaux	Examiner
Jérôme LEROUX	Directeur de Recherche	Université de Bordeaux	President
Laure PETRUCCI	Professeur des Universités	Université Paris 13	Reviewer
B Srivathsan	Associate Professor	CMI, India	Codirector
Igor WALUKIEWICZ	Directeur de Recherche	Université de Bordeaux	Director

To parents

Titre : Réduction d'ordre partiel pour les systèmes temporisés

Résumé : Dans cette thèse, nous étudions le problème d'accessibilité dans les réseaux d'automates temporisés. Nous nous concentrons sur la question de l'explosion de l'espace d'états causée par les entrelacements d'actions dans les exécutions de ces réseaux. Nous proposons deux solutions différentes pour atténuer l'effet de cette explosion combinatoire. La première est un algorithme d'accessibilité basé sur une sémantique temporelle locale pour les réseaux d'automates temporisés. La seconde est un cadre pour les méthodes de réduction d'ordre partiel pour ces mêmes réseaux.

L'approche standard pour résoudre le problème d'accessibilité d'un automate temporisé implique l'exploration d'un graphe dirigé, connu sous le nom de graphe de zone de cet automate. Dans notre travail, nous considérons une sémantique alternative pour les réseaux d'automates temporisés, appelée sémantique temporelle locale, qui a été introduite par Bengtsson et al [BJLY98], ainsi que la notion correspondante de graphes de zone locale. Cette nouvelle approche consiste à considérer une échelle de temps locale à chaque processus, et à ne synchroniser ces temps locaux que lorsque les processus exécutent une action commune. Le principal défi ici est que les graphes de zones locales sont infinis en général. Nous surmontons ce défi en concevant un nouvel algorithme qui vérifie l'accessibilité dans un réseau en calculant une troncature finie de son graphe de zone locale. Nous montrons que plusieurs nœuds dans le graphe de zone standard, qui correspondent à différents entrelacements des mêmes actions, sont remplacés par un seul nœud dans le graphe de zone locale. L'évaluation expérimentale de notre algorithme montre le gain d'un ordre de grandeur par rapport aux algorithmes de référence sur plusieurs exemples standards.

Dans la deuxième partie de cette thèse, nous nous concentrons sur l'application des techniques de réduction d'ordre partiel à l'exploration des graphes de zones locales. La réduction d'ordre partiel est une technique largement utilisée pour combattre l'explosion combinatoire de l'espace d'états. Elle consiste à identifier une petite partie de l'espace d'états dont l'exploration est suffisante pour vérifier le système. Nous décrivons pourquoi ceci est difficile à réaliser dans un cadre temporisé. Nous identifions ensuite une sous-classe de réseaux d'automates temporisés, que nous appelons systèmes à désynchronisation bornée, pour lesquels nous développons des algorithmes de réduction d'ordre partiel. Nous présentons plusieurs exemples pour cette sous-classe, motivés par des benchmarks standards. Nous fournissons également une évaluation d'un prototype de l'implémentation de ces méthodes en utilisant l'outil TChecker [HP19].

Mots-clés : Automates temporisés, Vérification, Réduction d'ordre partiel, Explosion de l'espace d'état

Title: Partial order reduction for timed systems

Abstract: In this thesis, we study the reachability problem for networks of timed automata. We focus on the issue of state-space explosion due to interleavings in these networks, and provide two different solutions for alleviating the effects of this explosion. The first is a reachability algorithm based on a *local time semantics* for networks of timed automata, and the second is a framework for *partial order reduction* methods for networks of timed automata.

The standard approach for solving the reachability problem for a timed automaton involves exploring a directed graph, known as the *zone graph* of the timed automaton. In our work, we consider an alternate semantics for networks of timed automata, called *local time semantics*, which was introduced by Bengtsson et al. [BJLY98], and the related notion of *local zone graphs*. The approach in local time semantics is to make time local to each process, and synchronize the local times of processes when they execute a common action. The main challenge here is that local zone graphs are infinite in general. We overcome this challenge by designing a new algorithm that checks reachability in a network by computing a finite truncation of the local zone graph of the network. We show that multiple nodes in the standard zone graph that correspond to different interleavings of the same sequence are replaced by a single node in the local zone graph. Experimental evaluation of our algorithm shows an order of magnitude gain with respect to state of the art algorithms on several standard benchmark examples.

In the second part of this thesis, we shift our focus to applying partial order reduction techniques to the exploration of local zone graphs. Partial order reduction is a widely used technique that combats the combinatorial explosion of search space by identifying a small part of the state space whose exploration is sufficient to verify the system. We describe why this is difficult to achieve in the timed setting. We then identify a subclass of networks of timed automata which we call *bounded spread systems* for which we develop partial order reduction algorithms. We exhibit several examples motivated by standard benchmark models that belong to this subclass. We also provide an evaluation of a prototype of the implementation of these methods using the tool TChecker [HP19].

Keywords: Timed automata, Verification, Partial order reduction, State-space explosion, Local time semantics

---

Unité de recherche
Laboratoire Bordelais de Recherche en Informatique, CNRS UMR 5800,
Université de Bordeaux, 33405 Bordeaux, France.
Chennai Mathematical Institute, Chennai, India.

Acknowledgements

I would like to first thank my advisors Igor, Frédéric and Srivathsan. The work in this thesis would not have been possible without their constant guidance, support and encouragement.

Both Igor and Frédéric have taken pains to ensure that I am comfortable during my stay at Bordeaux, especially during the pandemic. I am grateful to them for their constant care and concern. Throughout the course of my PhD, I have had the opportunity to learn a lot from them, starting from their way of approaching a problem to writing simple and elegant proofs. I admire their approach and outlook to research. I feel fortunate to be advised by Igor and Frédéric and express my deepest gratitude for their mentorship.

I am also grateful to Igor for patiently guiding me with the writing of this thesis - his valuable comments and advice have been vital in vastly improving it. The prototype of our algorithms in the tool TChecker is thanks to the efforts of Frédéric, and I thank him for all his hard work towards incorporating our results in TChecker. I would also like to thank Frédéric for going through the thesis carefully, and giving several valuable comments and suggestions that have enhanced the presentation of the thesis. Thanks to Frédéric for helping me with French translations and helping me navigate the bureaucratic system at Bordeaux.

I have learnt a lot from the several long and enjoyable discussions with Srivathsan throughout the last 3 years. It was Srivathsan who introduced me to timed automata during my Masters at CMI. His approach of stripping the problem to its bare basics and then attacking it has helped me in my work. He has always been ready to help and advise me in times of need, and I could not have asked for a better mentor.

I would like to thank the reviewers of this thesis, Laure Petrucci and Thomas Chatain for for taking the time to read this thesis, for evaluating my work, and for providing valuable feedback. I would also like to thank the reviewers and the examiners, Jérôme Leroux, Silvano Dal Zilio and David Ilcinkas for the interesting questions and perspectives during the defense.

I would like to thank UMI Relax for funding this cotutelle thesis and extend my thanks to Madhavan Mukund and Pascal Weil for giving me this opportunity. I would also like to thank Pascal for the several interesting lunch discussions during my stay at Bordeaux. I would like to express my

gratitude to the members of my Comité de suivi, Yves Metivier and Diego Figuera, for their advice and guidance.

I take this opportunity to thank the administrative staff at LaBRI, University of Bordeaux and CMI for their help with all of the administrative formalities.

I would like to thank my officemates Quentin, Jonathan, Jana and Noureddine for making Office 123 my home away from home. Special thanks to Marthe for feeding us with cookies from time to time! Many thanks to Quentin and Christele for hosting the 'traditional' Board game nights that we all enjoyed a lot.

I would also like to thank Sougata, Soumyajit, Kanka, Mallika, Aline, Debam, Varun, Attila, Raj, Ritam, Jay, Ani, Karim 1 and 2, Trang, Josephine, Shih-shun, Tidiane, Sidoine, Roopayan and many others for making life in Bordeaux memorable. I would like to thank my teachers at CMI, IMSc and NIT Calicut who introducing me to various areas of computer science. I am grateful to Amaldev Manuel and Paul Gastin for introducing me to research in Automata theory, logics and algebra during my Masters at CMI. I have learnt a lot from both of them and I thank both of them for working with me, teaching me several lessons in research and for the great discussions we had.

I would like to thank my friends, Nithin, Pranav, Vishnu, Sayan, Ashish, Krishnendu, Rajit, for their support and encouragement during the writing of the thesis. I am grateful to Nithin for his support and valuable suggestions at various stages of my work. Thanks also to Sayan, Pranav and Ashish for providing me with detailed feedback about the manuscript at various points. I would also like to thank all my friends from CMI and NIT Calicut who have been an integral part of my life.

I was introduced to computer science by Prof. Murali Krishnan during my bachelors at NIT Calicut. I owe him a deep debt of gratitude for all I have learned from him, and for all his support, guidance and mentorship.

I am deeply indebted to Prof. A G Menon for always being there to advise me and for all his kind words of encouragement ever since I first met him as a small boy at IISc. I would also like to thank Prof. Vinod A P for his advice and encouragement.

I would like to thank all my friends and relatives for their support and encouragement. I will not list them, but I would like to express my gratitude to each and everyone of them for their constant support.

Lastly, I thank my parents, my sister and my grandmother for all their unconditional love and support. My parents have always done everything possible to help me pursue my interests. They are a source of inspiration for me and if I have accomplished anything, I owe it to them.

Résumé de la thèse

Les systèmes cyber-physiques sont largement utilisés dans divers domaines tels que l'industrie automobile, la santé, l'avionique et l'industrie manufacturière. Des contrôleurs industriels aux stimulateurs cardiaques, en passant par les systèmes automobiles autonomes, ils sont omniprésents dans la société actuelle et jouent un rôle central dans l'assistance et l'amélioration de divers aspects de notre vie quotidienne. Ces systèmes ont généralement des spécifications, exprimées sous forme de contraintes sur leur comportement. Très souvent, ces systèmes sont essentiels à la sécurité ; même une chance infime de comportement erroné entraînerait une perte de vie ou de biens. Il est donc crucial de s'assurer que ces systèmes répondent toujours à leurs spécifications.

La vérification formelle est un domaine de recherche consacré à l'élaboration de procédures permettant de résoudre ce problème. La vérification des systèmes cyber-physiques consiste à modéliser les composants du système en tant qu'objets mathématiques abstraits, et à concevoir des algorithmes permettant de vérifier que le comportement de ces systèmes est conforme aux attentes. Les *automates temporisés* sont un formalisme pratique, utilisé lorsque les systèmes à modéliser sont associés à des contraintes temporelles. Il est souvent plus naturel de modéliser ces systèmes par un réseau d'automates temporisés qui fonctionnent simultanément et se synchronisent sur des actions communes.

L'explosion de l'espace d'états est un défi important dans la conception de procédures de vérification pour les réseaux d'automates. Cette explosion fait référence à la croissance exponentielle de la taille de l'espace d'états du réseau lorsque la taille du réseau augmente. Nous montrons que si l'explosion de l'espace d'états est déjà un défi pour les réseaux d'automates non temporisés, cela empire dans le cas des automates temporisés, car chaque entrelacement stocke l'information temporelle et conduit à un état temporisé potentiellement différent.

Les *réductions d'ordre partiel* sont une famille de techniques employées pour résoudre le problème de l'explosion combinatoire en réduisant l'espace d'états à parcourir par un algorithme de vérification. Son application aux réseaux de systèmes non temporisés est bien étudiée [Val89, God90, Pel93, AAJS17], mais son transfert à des systèmes temporisés reste un défi [BJLY98,

DGKK98, HLL+14].

Cette thèse vise à développer les fondements des méthodes d'ordre partiel pour les réseaux d'automates temporisés, afin d'obtenir des solutions algorithmiques évolutives et des outils pour leur vérification. Nous considérons la sémantique temporelle locale [BJLY98] pour les réseaux d'automates temporisés et travaillons avec le graphe de zones locales, le graphe de zones dans cette sémantique. L'obstacle majeur lorsque l'on travaille avec des graphes de zones locales est qu'ils sont infinis en général. Nous proposons des techniques pour obtenir des troncatures finies de ces graphes de zones locales. Grâce à ces techniques, nous sommes en mesure de développer un cadre qui permet l'application de techniques d'ordre partiel à l'exploration du graphe de zone local de ces réseaux. Nous identifions ensuite certaines classes de réseaux d'automates temporisés et proposons un algorithme de réduction d'ordre partiel pour les réseaux appartenant à ces classes. Nous fournissons également une évaluation d'un prototype de l'algorithme de réduction d'ordre partiel sur quelques exemples en utilisant l'outil TChecker [HP19].

Problème d'accessibilité pour les automates temporisés

Nos principaux objets d'intérêt sont les réseaux d'automates temporisés. Un *automate temporisé* [AD90, AD94] est un automate fini dont les états sont équipés d'un ensemble de variables non négatives à valeurs réelles appelées *horloges*. Les horloges sont initialement fixées à zéro et augmentent à la même vitesse. Les transitions de l'automate sont associées à une conjonction de contraintes d'horloge, appelées *gardes*, où chaque clause de la conjonction implique la comparaison d'une horloge avec un nombre naturel. D'un point de vue opérationnel, cela signifie que la transition ne peut être prise que si les valeurs des horloges satisfont la garde associée à la transition. De plus, certaines horloges peuvent être remises à zéro lors de la prise d'une transition. De plus, un automate temporisé possède également un état initial et un ensemble d'états acceptants. Le comportement erroné du système est modélisé comme l'atteinte d'un état acceptant de l'automate temporisé. Le problème d'accessibilité d'un automate temporisé demande s'il existe une exécution de l'automate d'un état initial à un état acceptant. Par conséquent, vérifier si le système a une exécution erronée se traduit par le problème d'accessibilité de l'automate temporisé qui modélise le système. Les algorithmes de vérification pour les automates temporisés ont été largement étudiés et sont à la base de nombreux outils de vérification tels que TChecker [HP19], UPPAAL [LPY97, BDL+06], KRONOS [BDM+98], HYTECH [HHW97], LTSmin [KLM+15], CMC [LL98], PAT [SLDP09] and Theta [THV+17].

L'un des principaux défis rencontrés dans l'étude du *problème de l'acces-*

sibilité provient du fait que l'espace des valuations d'un automate temporisé est de taille infinie indénombrable. L'approche standard pour vérifier l'accessibilité des automates temporisés est basée sur l'exploration d'un graphe dirigé appelé *graphe de zone* de l'automate temporisé [DT98] dont les nœuds sont des paires (état, zone) constituées d'un état de l'automate et d'une zone [BY03]. Une *zone* est un ensemble de valeurs qui être représentée efficacement en utilisant des contraintes de différence entre horloges. Il a été démontré [DT98] qu'un automate temporisé A possède une exécution acceptante si et seulement s'il existe un chemin dans le graphe de zones de A vers un nœud avec un état acceptant. Ainsi, le problème d'accessibilité de l'automate temporisé se résume maintenant à la recherche d'un nœud avec un état acceptant dans le graphe de zone de A.

Cependant, cette approche comporte un piège : le graphe de zones peut être infini et, par conséquent, un algorithme explorant le graphe de zones peut ne pas terminer. Diverses *techniques d'abstraction* permettant de calculer des troncatures finies des graphes de zones, qui préservent les exécutions de l'automate, ont été étudiées [DT98, BBLP06, HSW12, GMS19].

L'approche du graphe de zone fonctionne bien pour les réseaux d'automates temporisés contenant un petit nombre de composantes. Cependant, lorsque le nombre de composantes augmentent, nous sommes confrontés au problème de l'explosion de l'espace d'états. Nous soulignons que si l'explosion de l'espace d'état est déjà un défi pour les réseaux d'automates non temporisés, elle l'est encore plus pour les réseaux d'automates temporisés. Dans un réseau de processus (non temporisés), les différents entrelacements d'un même ensemble d'actions indépendantes conduisent au même état. Ce n'est pas le cas dans les graphes de zones des réseaux d'automates temporisés, car les informations de temporisation (telles que l'ordre de remise à zéro des horloges) sont stockées dans ces zones. En conséquence, pour un ensemble d'actions donné, chaque entrelacement conduit à un nœud potentiellement différent du graphe de zones. Par conséquent, l'algorithme d'exploration du graphe de zones ne s'adapte pas bien lorsque le nombre de composantes du réseau augmente.

Notre travail

Dans cette thèse, nous considérons le problème de l'explosion de l'espace d'états dû aux entrelacements dans les réseaux d'automates temporisés, et la thèse contribue aux fondements des méthodes d'ordre partiel pour ces réseaux. Dans notre travail, nous considérons la *sémantique temporelle locale* pour les réseaux d'automates temporisés, qui est une sémantique alternative pour ces réseaux introduite par Bengtsson et al. [BJLY98]. La sémantique temporelle locale est basée sur une idée nouvelle et élégante : faire progresser

le temps dans chaque processus de manière indépendante, et synchroniser les temps locaux des processus lorsqu'ils doivent effectuer une action commune. Dans la sémantique standard, puisque l'écoulement du temps est toujours synchronisé entre tous les processus du réseau, il existe une notion de temps global. En revanche, dans la sémantique temporelle locale, chaque processus a sa propre horloge de référence qui suit son temps local. Nous travaillons avec des valuations locales, des zones locales et des graphes de zones locales, qui sont analogues aux valuations standard, aux zones standard et aux graphes de zones standard, respectivement. Dans le cadre de notre travail, nous proposons deux solutions différentes pour relever le défi de l'explosion de l'espace d'états pour les réseaux d'automates temporisés.

Sémantique temporelle locale et test d'accessibilité efficace pour les réseaux d'automates temporisés

Dans la première partie de la thèse, nous montrons quelques propriétés des graphes de zones locales. En utilisant ces propriétés, l'exploration du graphe de zones locales d'un réseau s'avère être une alternative plus attrayante que l'exploration du graphe de zones standard. Cependant, cette approche est entravée par l'absence d'un mécanisme permettant de garantir la finitude du graphe de zones locales. Cela crée le besoin d'une technique d'abstraction (similaire aux techniques d'abstraction pour les zones standard) pour les zones locales afin de rendre le graphe des zones locales fini.

Bengtsson et al. [BJLY98] et Minea [Min99a] ont proposé des techniques d'abstraction pour les zones locales. Nous montrons que chacune des solutions proposées présente certains défauts qui limitent leur applicabilité. Ainsi, il n'existait pas de moyen efficace d'utiliser la sémantique temporelle locale. Nous résolvons ce problème en introduisant une abstraction pour les zones locales, appelée *sync-subsumption*. Grâce à cette abstraction, nous sommes en mesure de calculer un système de transition fini appelé le *graphe local sync* qui est en général plus petit que le graphe de zone standard qui est exploré dans l'algorithme d'accessibilité standard. Nous proposons un nouvel algorithme d'accessibilité pour les réseaux d'automates temporisés qui est basé sur l'exploration du "graphe local sync" du réseau. Nous présentons également les résultats expérimentaux de l'application de l'algorithme à quelques exemples en utilisant un prototype implémenté dans l'outil TChecker [HP19]. Notre algorithme est étonnamment performant sur certains exemples, et sur aucun exemple, il n'est pire que l'algorithme standard utilisant le graphe de zone.

Fondements de la réduction de l'ordre partiel pour les réseaux d'automates temporisés

Dans la deuxième partie de cette thèse, nous proposons un algorithme de réduction d'ordre partiel pour le problème d'accessibilité d'une sous-classe de

réseaux d'automates temporisés. Nous discutons d'abord l'application d'une méthode existante de réduction d'ordre partiel au "graphe local sync" et expliquons pourquoi cela n'est pas correct. Par conséquent, si nous voulons appliquer la réduction de l'ordre partiel à l'exploration des graphes de zones locales des réseaux d'automates temporisés, nous avons besoin d'une autre version finie des graphes de zones locales qui répondent aux exigences de la réduction de l'ordre partiel.

À cette fin, nous proposons une nouvelle abstraction pour les zones locales, nommée $\mathfrak{a}^*_M$, qui est basée sur une relation de simulation pour les valeurs locales. En appliquant l'abstraction $\mathfrak{a}^*_M$ aux graphes de zones locales, nous obtenons un graphe qui se prête à une réduction d'ordre partielle. Cependant, nous montrons que l'abstraction $\mathfrak{a}^*_M$ n'est pas finie en général. Nous observons que la difficulté critique pour obtenir des abstractions finies de graphes de zones locales réside dans la divergence arbitraire entre les horloges de référence de différents processus. Garder la trace des différences arbitrairement grandes entre les horloges de référence constituent une difficulté majeure inhérente au travail avec des zones locales. Pour préciser cette idée, nous introduisons le concept de *spread* d'une valuation, qui est défini comme la différence maximale entre deux horloges de référence dans la valuation. Nous limiterons notre attention à une sous-classe de réseaux d'automates temporisés, que nous appellerons les *systèmes à désynchronisation bornée*, c'est-à-dire les réseaux pour lesquels, étant donné toute séquence d'actions réalisable dans son graphe local, il est possible de trouver une exécution où la différence entre les horloges de référence est bornée à tout moment. Nous proposons également quelques conditions pour vérifier si un réseau d'automates temporisés est à désynchronisation bornée.

Nous considérons une abstraction modifiée, nommée $\mathfrak{a}^D_M$, et qui correspond à l'abstraction $\mathfrak{a}^*_M$ paramétrée par une constante D. Nous montrons que si un réseau est à désynchronisation bornée par une borne D, l'application de l'abstraction $\mathfrak{a}^D_M$ tout en explorant le graphe de zone locale produit un graphe fini qui est correct pour l'accessibilité, et qui se prête à une réduction d'ordre partielle.

Nous identifions ensuite deux classes de réseaux d'automates temporisés, que nous appelons respectivement *systèmes global-local* et *systèmes client-serveur*, et nous proposons des procédures spécialisées de réduction d'ordre partiel pour les réseaux à limites étendues appartenant à ces catégories. Enfin, nous fournissons une évaluation d'un prototype sur quelques exemples en utilisant l'outil TChecker [HP19].

Contents

Chapter 1

Introduction

Cyber-physical systems are used extensively in various fields such as automotive industries, healthcare, avionics, and manufacturing. From industrial controllers to pacemakers and autonomous automobile systems, they are ubiquitous in today's society and have central roles in assisting and improving various aspects of our daily life.

Cyber-physical systems are usually associated with some constraints, and the requirements to satisfy these constraints are often very strict. In many cases, the restrictions on the behaviour of these systems are expressed in terms of time constraints. In other words, these systems may be expected to perform actions respecting certain time deadlines. Consider the example of a pacemaker. A pacemaker monitors the timing signals that it receives from sensors planted in the heart. If the pacemaker detects a timing pattern symptomatic of arrhythmia at any point of time, it is expected to generate electrical impulses. These signals are to be delivered by electrodes to the heart muscles, which causes them to contract and thereby pump blood. It is crucial that whenever a bad pattern in signals is detected, the pacemaker acts on it within a specified time window. Even a remote chance of erroneous behaviour or malfunctioning incurs a loss of human life and property. Therefore, it is vital to have some mechanism to ensure the correct behaviour of these systems at all times, so much so that this is tantamount to ensuring the smooth functioning of our society.

Formal verification is a research area that attempts to ensure the correct behaviour of systems by providing automated algorithmic solutions. Specifically, cyber-physical systems are modelled using abstract mathematical objects and algorithms are designed to verify that the behaviour of such systems is as desired. *Timed automata* are a convenient formalism used in situations where the systems we want to model have associated temporal constraints. It is often more natural to model these systems as a network of timed automata that operate concurrently and synchronize on joint actions.

State-space explosion is a significant challenge in designing verification

algorithms for complex systems modelled by networks containing a large number of components. In fact, in practice, a lot of popular cyber-physical systems are modelled using large networks. Hence, the scalability of algorithmic solutions is as critical as their correctness and efficiency.

Partial order reduction refers to a family of techniques employed to address the issue of state-space explosion by reducing the state space to be searched by a verification algorithm. Its application to networks of untimed systems is well studied [Val89, God90, Pel93, AAJS17], but their transfer to timed setting [BJLY98, DGKK98, HLL$^+$14] remains a challenge.

The partial order techniques crucially depend on an effective way to compute the *independence relation* between actions. While this is relatively easy to compute for untimed systems, in timed systems, since there is always an implicit synchronization between the components of the network because of time, it is not easy to compute the independence relation between actions of networks of timed automata. This constitutes the principal difficulty in extending partial order reduction techniques to networks of timed automata.

This thesis aims to develop foundations for partial order methods for networks of timed automata, so as to obtain scalable algorithmic solutions and tools for their verification. We consider the local time semantics [BJLY98] for networks of timed automata and work with the local zone graph, the zone graph in this semantics. The major obstacle while working with local zone graphs is that they are infinite in general. We propose techniques to obtain finite truncations of these local zone graphs. Thanks to these techniques, we are able to develop a framework that allows the application of partial order techniques to the exploration of the local zone graph of these networks. We then identify some classes of networks of timed automata and propose a partial order reduction algorithm for networks belonging to these classes. We also provide an evaluation of a prototype of the partial order reduction algorithm on some examples using the tool TChecker [HP19].

1.1. Reachability problem for timed automata

The primary objects of our interest are networks of timed automata. A *timed automaton* [AD90, AD94] is a finite state automaton equipped with a set of non-negative real-valued variables called *clocks*. The clocks are initially set to zero and increase at the same rate. The transitions of the automaton are associated with a conjunction of clock constraints, referred to as *guards*, where each clause in the conjunction involves the comparison of a clock with a natural number. Operationally, this means that the transition can be taken only if the value of the clocks satisfies the guard associated with the transition. In addition, some clocks can be *reset* (i.e., their values can be set to 0) on taking a transition. Moreover, a timed automaton also has an

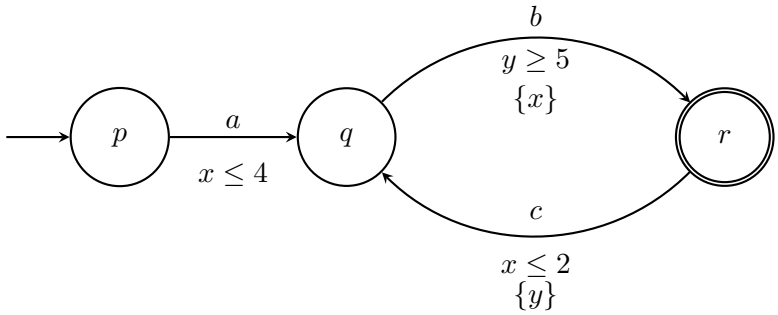


Figure 1.1: A timed automaton A. The guard associated with a transition is given under the transition. We write $\{x\}$ under the transition b to denote that the clock x is reset by b.

initial state and a set of accepting states. Figure 1.1 gives an example of a timed automaton with clocks $\{x, y\}$, with initial state p and accepting state r, over the alphabet $\{a, b, c\}$.

The clock variables allow us to measure the timing information associated to executions of the timed automaton and restrict and control its behaviour. For instance, we can measure the time elapsed between the execution of two transitions, or enforce that a transition is executed only after a specified amount of time.

A fundamental problem arising in the study of timed automata is the *reachability problem*. The reachability problem for a timed automaton asks if there exists a run of the automaton from an initial state to an accepting state. As mentioned earlier, timed automata are used to model cyber-physical systems. The erroneous behaviour of such a system can be modelled as reaching a special "bad" state of the timed automaton modelling it. Therefore, checking whether the system has an erroneous execution translates to checking the reachability of the timed automaton. Verification algorithms for timed automata have been widely studied and are the basis for a number of verification tools such as TChecker [HP19], UPPAAL [LPY97, BDL+06], KRONOS [BDM+98], HYTECH [HHW97], LTSmin [KLM+15], CMC [LL98], PAT [SLDP09] and Theta [THV+17].

One of the main challenges encountered in the study of the reachability problem stems from the fact that the space of valuations of a timed automaton is uncountably infinite. Recall that a timed automaton is a deterministic finite state automaton (DFA) whose transitions are associated with guards involving clock variables. For any fixed sequence of transitions in the underlying DFA, we have uncountably many executions of the timed automaton, where each execution differs from the other in the assignment to the clock variables (valuations) in at least one of the steps of the execution. We illustrate this in Figure 1.2, where we show some runs of the timed automaton from Figure 1.1

for the sequence $ab\cdots$.

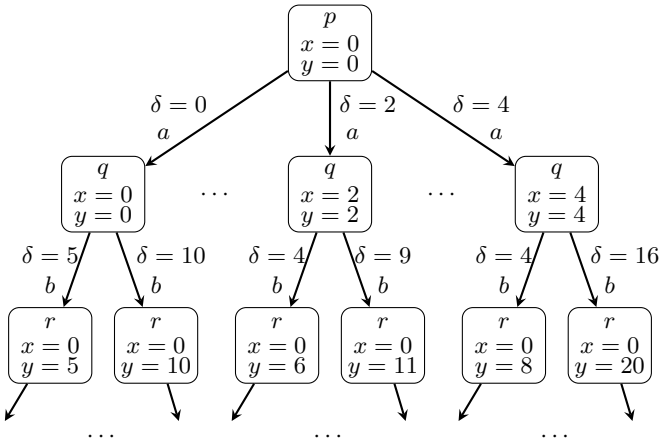


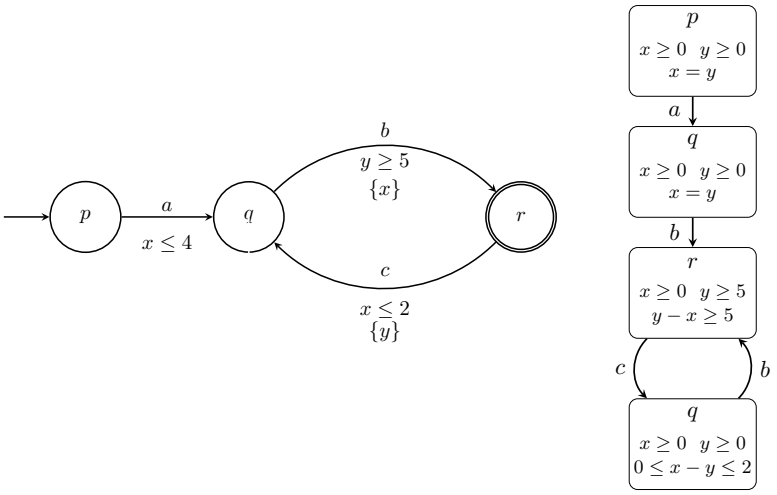
Figure 1.2: Uncountably infinite space of executions of the timed automaton A. $\delta = 2$ in an execution denotes a time elapse of 2 time units in the respective state, after which a transition is taken. Observe that the state r is associated with an uncountably infinite set of valuations. As a consequence, we have an uncountably infinite set of executions of A.

To tackle this problem, we first need a way to handle this uncountably infinite set of valuations effectively. The standard approach to overcome this involves using a symbolic representation to analyse timed automata.

One of the earliest solutions to deal with the uncountably infinite set of valuations was proposed by Alur and Dill, presented in [AD94]. Their idea is to partition the set of valuations into a finite number of sets, called *regions*. Regions are essentially sets of valuations that are indistinguishable by any timed automaton. This means that if a run is feasible from a valuation in a region, then it is also feasible from any valuation in the region. Equipped with this notion of regions, Alur and Dill propose a transition system called the *region graph* of the timed automaton. Exploring the region graph of a timed automaton is a valid way to solve the reachability problem; however, in practice, it turns out that the region graphs are too large to be of use for efficiently checking the reachability of timed automata.

The most widely used approach for checking reachability in timed automata is based on the notion of *zones* [BY03], which are sets of valuations that can be represented efficiently using difference constraints between clocks. Consider a timed automaton A with initial state q_0 and initial valuation v_0. If we consider the execution of a specific sequence of transitions from (q_0, v_0), we end up with a state whose associated set of valuations is uncountably infinite. It turns out that for any sequence of transitions, this set of valuations can be expressed as a zone. In particular, we can associate a state and a zone to each sequence of transitions. This idea is used to construct a *zone*

graph of the timed automaton [DT98] whose nodes are (state, zone) pairs consisting of a state of the automaton and a zone representing a set of clock valuations [Dil89]. Figure 1.3 gives a timed automaton and its zone graph. It was also shown [DT98] that a timed automaton A has an accepting run if



(a) A timed automaton A (b) Zone graph of A

Figure 1.3: A timed automaton and its zone graph

and only if there is a path in the zone graph of A to a node with an accepting state. Thus, the reachability problem of the timed automaton now amounts to a search for a node with an accepting state in the zone graph.

However, this approach has a catch - the zone graph could be infinite, and therefore, an algorithm exploring the zone graph might not terminate. Various sound and complete *abstraction* techniques that allow us to compute finite truncations of zone graphs have been widely studied, and this area still witnesses active research [DT98, BBLP06, HSW12, GMS19]. One way to get an abstraction is to make use of a *subsumption relation* on zones. Roughly, what this means is that if a zone Z is subsumed by another zone Z', then every sequence of actions that is feasible from (q, Z) is also feasible from (q, Z'). Hence, an algorithm exploring the zone graph does not need to explore paths from (q, Z), and only needs to explore from (q, Z'). Termination is guaranteed if the subsumption relation induces a finite number of maximal zones.

The zone graph approach works well for networks of timed automata containing a small number of components. However, as the number of participating components increases, we are faced with the problem of *state-space explosion*, which we discuss next.

1.2. State-space explosion

State-space explosion refers to the exponential growth in the size of the transition system describing the semantics of a network of processes due to multiple interleavings of the same set of transitions. We will now illustrate this phenomenon with the help of an example. Consider a network of n processes, each having k states. The size of the resultant product automaton grows exponentially with respect to n. In particular, if the network is such that there are no synchronizations between the participating components, then the resultant product automaton has k^n states. Further, if we consider a sequence of n actions, each belonging to a different process, then there are $n!$ different interleavings of this sequence, one per each possible ordering of these actions. We give an example of a network of three (untimed) processes and its product automaton in Figure 1.4. Note that 3! paths go from the initial state (p_0, q_0, r_0) to the state (p_1, q_1, r_1). These paths correspond to the transitions a_1, b_1 and c_1 executed in different orders.

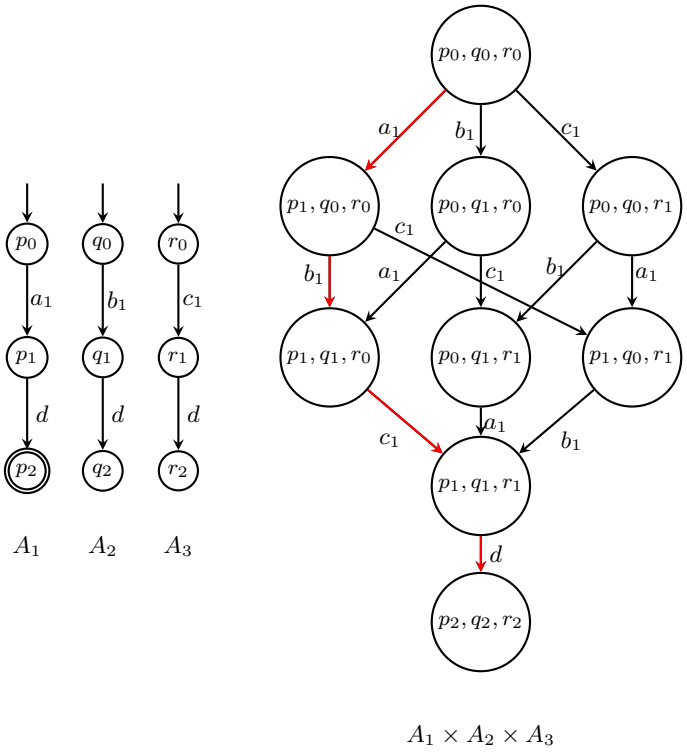


Figure 1.4: A network of three (untimed) processes

In the product automaton of a network of (untimed) processes, different interleavings of the same sequence lead to the same state. Unfortunately, this is not the case in zone graphs of networks of timed automata as timing

information (such as the order of reset of clocks) is stored in these zones.

Consider a simple network of timed automata as given in Figure 1.5. Variables x, y are clocks, and $\{x\}$ denotes the reset of clock x. As mentioned earlier, the zone graph maintains the resultant (state, zone) pairs for each sequence of transitions. Hence, the zone reached after executing the sequence ab contains configurations where x is reset before y, while the zone reached on executing ba contains configurations where y is reset before x. This implies that the valuations of the former zone satisfy the constraint $x \leq y$, while the valuations of the latter zone satisfy the constraint $y \leq x$. Thus, the sequences ab and ba lead to different zones. Note that in an untimed version of this network, we would have only one instance of the state (p_1, q_1) in the product automaton. On the other hand, in the zone graph, we have two nodes with the state (p_1, q_1), one per each possible ordering of clocks $\{x, y\}$. This indicates that the problem of state-space explosion is much more severe for networks of timed automata. The number of different interleavings of

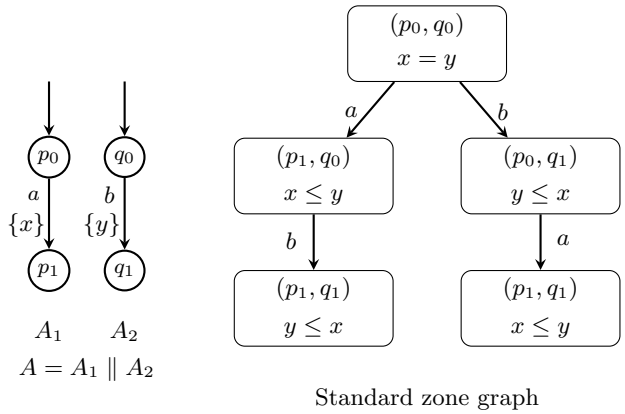


Figure 1.5: A network A and its zone graph

sequences of independent transitions increases exponentially with the number of components in the network. Consequently, the zone graph exploration algorithm does not scale well when the number of components of the network increases.

1.3. Partial order reduction

Partial order reduction (POR) is a term used to describe the broad class of techniques used for reducing the state-space of the transition system to be explored by a verification procedure [CGMP99, Val89, Pel93, God96, FG05, AAJS17]. It is one of the standard approaches used to tackle the problem of state-space explosion for systems consisting of several components operating

in parallel. Partial order reduction has proven quite useful in improving the performance, both in terms of running time as well as the memory requirement of verification procedures for such systems [GPS96, KLM+98]. Partial order reduction has also been implemented in the model checking tool SPIN [God96, HP94].

The general principle of partial order reduction techniques is the following: classify executions of a system into equivalence classes and explore only one representative from each equivalence class. Specifically, two executions that are interleavings of concurrent independent transitions are deemed to belong to the same equivalence class. This idea of grouping runs into equivalence classes on the basis of commuting pairs of adjacent independent transitions has its roots in the work of Mazurkiewicz [Maz86].

Consider the network of untimed automata given in Figure 1.4. Observe that if we are only interested in the reachability of the tuple (p_2, q_2, r_2), we do not need to explore the complete product transition system. In this regard, all paths from (p_0, q_0, r_0) to (p_2, q_2, r_2) are equivalent. So, it is sufficient to explore a smaller transition system, referred to as a *reduced transition system*. The part of the transition system $A_{\mathcal{N}}$ given by the red edges in Figure 1.4 is an example of a reduced transition system.

Next, we discuss how such a reduced transition system can be constructed. From the example in Figure 1.4, we can see that we are avoiding the exploration of multiple interleavings of actions a_1, b_1 and c_1, as all the interleavings lead to the same state - in other words, the order in which they are executed is irrelevant. We need to identify sets of actions such that the order of execution of these actions is irrelevant. We now formalize this notion.

Given a transition system T, we say that an action a is *enabled* from a state s in T, if there exists a transition $s \xrightarrow{a} s'$ in T. We say that two actions a and b are *independent* in a state s of T [KP92], if they satisfy the following conditions:

Forward diamond property If a and b are enabled from s, and $s \xrightarrow{a} s_1$ and $s \xrightarrow{b} s_2$ are the respective transitions from s, then b is enabled from s_1 and a is enabled from s_2.

Diamond property If either ab or ba is feasible from s, then both ab and ba are feasible from s and moreover, both ab and ba result in the same state. The diamond property is pictorially represented by the diamond structure given in Figure 1.6.

We say that two actions a and b are independent, if they are independent in each state s of T. From the definition above, we can see that if two actions a and b are independent in a state s, then both the sequences ab and ba are feasible from s, and both lead to the same state. As illustrated in Figure 1.6, if we are only interested in the reachability of state s', it is sufficient to explore only the action a from the state s. Since a and b are independent,

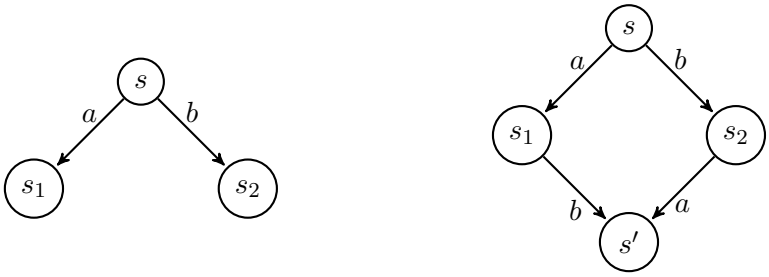


Figure 1.6: Diamond property

it is guaranteed that b will be enabled from the state s_1 reached on the execution of a from s. Effectively, we are postponing the execution of the action b, as we are sure that it will be enabled later.

Suppose that we have a method to check the independence of actions from a state. Then, from each state of the transition system, we can identify a subset of the set of enabled actions, such that it is sufficient to explore this set of actions. In other words, the requirement is that each action that is not explored is guaranteed to be enabled later. It is possible now to consider a *reduced transition system* that includes only those states that are reachable from the initial state via only those actions picked in the aforementioned subset of enabled actions from each state.

Clearly, this reduced transition system is *sound*, i.e., if there is an accepting run in the reduced transition system, then it is also a run in the original transition system. A crucial requirement is that this reduced transition system is also *complete*. In other words, if there is an accepting run in the original transition system, then there should also be an accepting run in the reduced transition system. One could easily compute such a reduced transition system after computing the full transition system and then looking at equivalent paths, but this defeats the whole purpose of partial order reduction. Consequently, the reduced transition system should be computed on the fly. An important condition here is that we must only use information, that is either readily available from a state or that we already know from the exploration of the system so far, to choose the subset of actions to be explored from that state.

To summarize, we want to compute a function that maps each state of the transition system to a subset of enabled actions such that

- the resultant reduced transition system is complete.
- this function is computed on the fly, using only local information from each state.

Observe that the trivial function that returns the set of all enabled actions for each state produces a transition system that satisfies the requirements

above. However, there is no reduction in terms of state-space or the number of transitions in the reduced transition system, and therefore, this function is useless. Hence, it is imperative to require that the function maps a state to a small subset of enabled actions from that state; the smaller this set is, the better the reduction observed.

1.4. Our work

This thesis contributes to the foundations of partial order methods for networks of timed automata. In our work, we consider the local time semantics for networks of timed automata from [BJLY98] and work with local valuations, local zones, and local zone graphs. These are analogous to standard valuations, standard zones, and standard zone graphs, respectively. As a part of our work, we propose two different solutions to tackle the challenge of state-space explosion for networks of timed automata.

In the first part of the thesis, we show some properties of local zone graphs. By making use of these properties, the exploration of the local zone graph of a network turns out to be a more appealing alternative than the exploration of the standard zone graph. However, this approach is hampered by the absence of a mechanism to ensure the finiteness of the local zone graph. We solve this problem by introducing a subsumption relation for local zones, referred to as *sync-subsumption*. Thanks to this subsumption, we are able to compute a finite transition system called the *local sync graph* which is in general smaller than the zone graph that is explored in the standard reachability algorithm. We propose a new reachability algorithm for networks of timed automata that is based on the exploration of the local sync graph of the network. We also present the experimental results of applying the algorithm to some examples using a prototype implemented in the tool TChecker [HP19]. This results of this part of the thesis appear in the paper [GHSW19].

The natural next step to further optimize the aforementioned algorithm is to apply a partial order reduction method while exploring the local sync graph. Unfortunately, we do not have an effective procedure to compute the independence relation of actions in local sync graphs. Therefore, if we are to apply partial order reduction to the exploration of local zone graphs of networks of timed automata, we need an alternate finite version of local zone graphs, for which we have an effective procedure to compute the independence relation.

To this end, we propose a new abstraction for local zones, $\mathfrak{a}_M^D$ *abstraction*, that is based on a simulation relation for local valuations. We also introduce the notion of *spread* of a network of timed automata and specify when a network of timed automata is *spread-bounded*. We show that if a network

is spread-bounded (with a bound D), applying the $\mathfrak{a}_M^D$ abstraction while exploring the local zone graph yields a finite transition system, called $\mathsf{LZG}_\mathsf{M}^\mathsf{D}$, that is sound and complete with respect to reachability. Moreover, we show that it is easy to compute an approximation of the independence relation between the actions of the $\mathsf{LZG}_\mathsf{M}^\mathsf{D}$ of a network of spread D.

We then identify two classes of networks of timed automata, which we refer to as *global-local systems* and *client-server systems* respectively, and propose specialized partial order reduction procedures for spread-bounded networks belonging to these categories. Finally, we provide an evaluation of a prototype on some examples using the tool TChecker [HP19].

1.4.1 Local time semantics and efficient reachability testing for networks of timed automata

For applying partial order reduction to a network of processes, it is crucial to have an effective way to compute the independence relation (see conditions given in page 8) between the actions of the network. For networks of untimed processes, independence of actions can be tested using a simple syntactic check: if two actions of the network have no common process participating in them, then they are independent. We refer to the set of processes participating in an action as the domain of the action. Further, we say that two actions have *disjoint domains* if they have no common process participating in them. Then, the check boils down to the following: if two actions have disjoint domains, then they are independent. Additionally, we say that two sequences of actions are equivalent if one can be obtained from the other by permuting adjacent actions with disjoint domains.

Observe that in the standard semantics for networks of timed automata, two actions having disjoint domains do not necessarily satisfy the conditions for independence. Consider the example of a network with two processes, as shown in Figure 1.7. In the network of timed automata $\mathcal{A}$, although a and b are actions local to their respective processes, there is an intrinsic dependence between the two processes, which arises due to implicit constraints imposed by time. Observe that network $\mathcal{A}$ can execute the sequence ab (sequence ab is feasible), while the sequence ba is not feasible. We do not know of any effective way to compute independence between actions in the standard semantics of networks of timed automata. There have been some attempts to come up with effectively checkable sufficient conditions for independence between actions in the standard semantics for networks of timed automata [DGKK98, HLL$^+$14]. The global nature of time elapse in a network of timed automata induces implicit dependencies between actions of different processes. This seems to be the major obstacle to getting effectively checkable conditions for independence between actions in networks of timed automata.

Local time semantics is an alternate semantics for networks of timed automata that addresses this very problem. Introduced by Bengtsson et

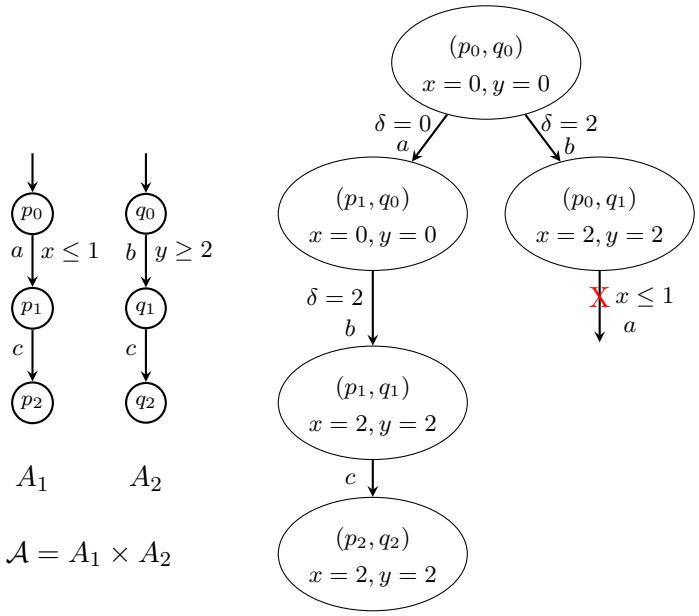


Figure 1.7: A network of timed automata and some runs in the standard semantics

al. [BJLY98], local time semantics is based on a novel and elegant idea: make time in each process progress independently, and synchronize local times of processes when they need to perform a common action. In the standard semantics, since time elapse is always synchronized across all processes of the network, there is a notion of global time. In contrast, in local time semantics, each process has its own *reference clock* that tracks its local time. Thus, for each process, we have a variable t_p which tracks the local time of the process: essentially t_p is a clock that is never reset. In local time semantics, instead of working with standard clocks, we work with an offset representation of clocks. For each clock x, we consider an offset variable $\widetilde{x}$ that stores the time-stamp at which x was last reset.

Observe that in the network from Figure 1.5, depending on the local time in processes A_1 and A_2, the sequence ab may result, on a global time scale, in a occurring before b, as well as b occurring before a. As a result, the set of valuations reached after the local run ab does not remember the order in which a and b occurred. Thus, sequences ab and ba lead to the same set of configurations: those obtained after doing a and b concurrently, as shown in Figure 1.8. It turns out that in local time semantics, two actions with disjoint domains satisfy the conditions for independence between actions (see conditions given in page 8). Further, we show that the local time semantics and the standard semantics are equivalent if we consider reachability with local valuations in which all the reference clocks are equal, referred to as

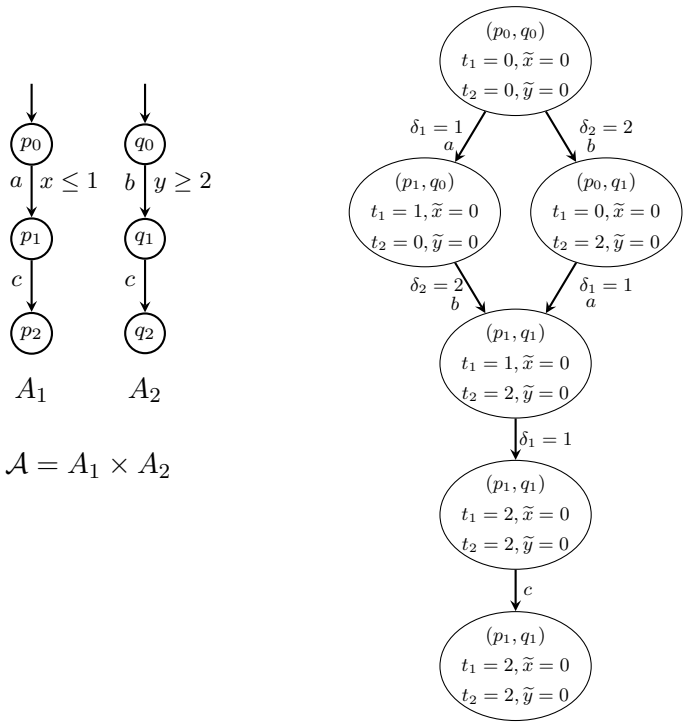


Figure 1.8: Runs in local time semantics. Here, $\widetilde{x}$ and $\widetilde{y}$ represent the offset clock variables corresponding to clocks x and y, respectively. The variables t_1 and t_2 denote the reference clocks of processes A_1 and A_2, respectively.

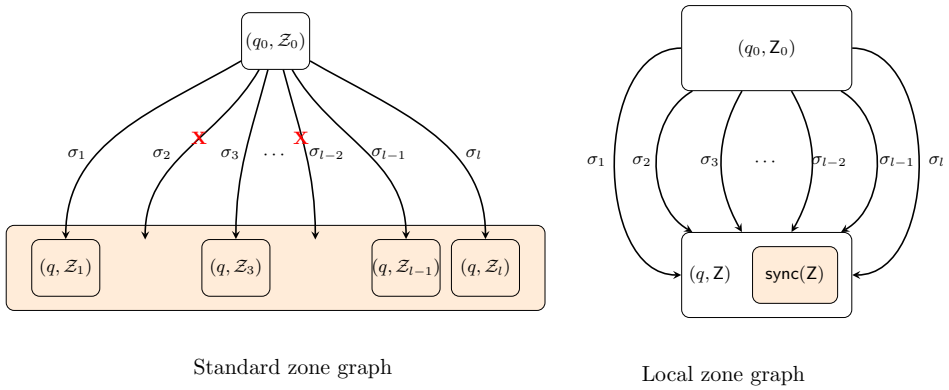
synchronized valuations. In other words, there is a run in the local time semantics to a state with a synchronized valuation if and only if there is a run in the global semantics to this state. Together, these facts make local time semantics a more apt setting to apply partial order reduction than the standard semantics.

Analogous to standard zones and zone graphs, in the setting of local time semantics, we have the notions of local zones and local zone graphs, whose nodes are pairs of the form (state, local zone). In local zone graphs, two actions with disjoint domains satisfy the conditions for independence between actions (see conditions given in page 8). We remark that two actions with disjoint domains satisfy only the diamond property and not the forward diamond property (see conditions given in page 8). Thus, we have a relaxed version of the condition for independence, which we show is still sufficient for applying partial order reduction. Further, it can be shown that the local zone graph of a network is sound and complete with respect to reachability. This means that instead of exploring the standard zone graph (which is the procedure used in the standard test for reachability) one could explore the local zone graph. Moreover, it can be shown that the local zone graph has the following very useful property: if a sequence of actions σ from a node (q, Z) of the local zone graph reaches the node (q', Z'), then all the sequences equivalent to σ are feasible from (q, Z); furthermore, they all lead to the same node (q', Z').

Equivalent sequences of actions in a network of timed automata behave quite differently in the standard zone graph and the local zone graph, as shown in Figure 1.9. Recall that two sequences of actions are equivalent if one can be obtained from the other by permuting adjacent actions with disjoint domains. Let σ be a sequence of actions from the initial node (q_0, Z_0) of the standard zone graph. Let $\{\sigma_1, \sigma_2, \cdots, \sigma_l\}$ be the set of all sequences equivalent to σ. Observe that not all of these sequences are feasible in the standard zone graph; for instance, in Figure 1.9, the sequences σ_2 and σ_{l-2} are not feasible in the standard zone graph. Even if they are feasible, these sequences lead to potentially different nodes in the standard zone graph. On the contrary, in the local zone graph, all the sequences equivalent to σ are feasible from (q_0, Z_0) and they all lead to the same node.

This ensures that if a sequence of actions σ is feasible in the local zone graph of a network, then all execution sequences equivalent to σ are also feasible. Local zone graphs are therefore ideal for handling interleavings.

Further, we show that this observation ties in neatly with a surprising property for standard zone graphs shown by Salah et al. [SBM06]: for any sequence of actions σ, the union of zones reached by the interleavings of σ is also a zone. For instance, in Figure 1.9, $\bigcup_{i=1}^{l} \mathcal{Z}_i$ is a zone. We refer to this zone as the *aggregated zone* of the sequence σ. Salah et al. [SBM06] argue that for a given sequence, one can write a zone-like constraint defining all the runs of the interleavings of the sequence. Then the aggregated zone is obtained



Standard zone graph

Local zone graph

Figure 1.9: Comparison of behaviour of equivalent sequences in the standard zone graph and the local zone graph

simply by projecting this big constraint on relevant components. Salah et al. use this observation in an algorithm where, when all the interleavings of a sequence σ have been explored, the resulting zones are aggregated to a single zone and further exploration is restricted to this aggregated zone.

We point out there are some obstacles in the approach by Salah et al. using aggregated zones to get efficient algorithms to check reachability. First, their approach requires one to work with sets of constraints whose size grows with the length of a sequence. This is both inefficient and limited to finite sequences. Secondly, this approach requires detecting from time to time whether aggregation can happen. Finally, another limitation of this approach is that it works only for acyclic automata.

We show that the aggregated zone of a sequence σ can be obtained by restricting the local zone Z reached on σ to synchronized valuations. We refer to the restriction of a local zone Z to only synchronized valuations as the *synchronized zone* of Z, denoted as $\mathsf{sync}(\mathsf{Z})$. Thus, we provide a way to automate this periodic detection of the completion of the exploration of all the interleavings of a sequence σ and the aggregation of the resulting zones. Therefore, an algorithm that answers the question of reachability by exploring the local zone graph appears to be a better solution.

However, we are faced with a major roadblock in this approach: local zone graphs are not finite in general, and therefore, an algorithm exploring a local zone graph may not terminate. This creates the need for an abstraction technique (similar to the abstraction techniques for standard zones) for local zones to make the local zone graph finite. Bengtsson et al. [BJLY98] and Minea [Min99a] have proposed abstraction techniques for local zones. We show that each of the proposed solutions has some flaws that restrict their applicability. In the paper that introduced local time semantics, Bengtsson et al. [BJLY98] defined a subsumption relation between local zones but did not

provide an algorithm to compute it. This meant that there was no effective way to use local time semantics. Later Minea [Min99a] proposed a widening operator on local zones to construct finite local zone graphs. We show that the solution in [Min99a] contains a bug, and it is not clear whether this approach can be fixed. In summary, up until now, there is no known way to compute finite local zone graphs.

We propose a new subsumption operator between local zones, referred to as *sync-subsumption*, denoted $\sqsubseteq^{\mathfrak{a}LU}_{\mathsf{sync}}$, that ensures termination of exploration of the local zone graph. We show that given a local zone Z, $\mathsf{sync}(\mathsf{Z})$ can be expressed as a standard zone. In sync-subsumption, we consider the containment of local zones with respect to synchronized valuations. It allows us to define the *local sync graph*, which is the local zone graph reduced by applying the sync-subsumption operation. We show that the local sync graph can be used to answer the reachability problem for networks of timed automata. This gives a new reachability algorithm for networks of timed automata that works with local zones but uses abstractions over standard zones. Recall that given a local zone Z, $\mathsf{sync}(\mathsf{Z})$ can be expressed as a standard zone. Consequently, the abstractions used in our algorithm are, in fact, abstractions over standard zones. This allows us to take advantage of the numerous improvements to abstraction operators over standard zones, developed over the years [DT98, BBLP06, HSW12, GMS19].

Thus, computing the local zone graph gives a more direct and efficient algorithm than Salah et al. to compute aggregated zones. Moreover, our algorithm is not restricted to acyclic timed automata.

To summarize,

- We adopt the local time semantics introduced by Bengtsson et al. [BJLY98] and develop a series of basic results about local valuations, local regions, and local zones.
- We point out a flaw in the abstraction procedure proposed by Minea [Min99a] to get a finite local zone graph.
- We propose a new subsumption technique called *sync-subsumption*, denoted as $\sqsubseteq^{\mathfrak{a}LU}_{\mathsf{sync}}$, to get a finite local zone graph. This gives a new reachability algorithm for networks of timed automata that works with local zones. To the best of our knowledge, this is the only reachability algorithm based on the exploration of the local zone graph. We observe that our new subsumption is much more aggressive than the standard $\mathfrak{a}_{\preccurlyeq LU}$ subsumption. Further, we show that this new way of computing the local zone graph gives a more direct and efficient algorithm than that of Salah et al. [SBM06] to compute aggregated zones.
- We report on experiments performed with a prototype implementation of the algorithm. The algorithm performs surprisingly well on some

examples, and on no example, it is worse than the standard zone graph algorithm.

1.4.2 Foundations for partial order reduction for networks of timed automata

In the second part of this thesis, we propose a partial order reduction algorithm for the reachability problem for networks of timed automata. As already discussed in Section 1.3, developing partial order methods for networks of untimed systems is well studied and has proven quite successful in optimizing the running time and memory consumption of verification procedures for these systems. Despite this huge success, their extension to networks of timed systems has largely remained a challenge.

Our first attempt to develop a partial order reduction algorithm for networks of timed automata involves applying an existing partial order reduction method to the algorithm based on the exploration of local zone graphs, which is introduced in the first part of the thesis. Recall that local zone graphs are not finite in general, and hence, we worked with local sync graphs, obtained by applying sync-subsumption to local zone graphs. However, we do not know how to compute the independence relation between the actions of a local sync graph. In particular, disjointness of domains, which was a sufficient condition for the independence of actions in local zone graphs, does not imply independence in local sync graphs. This seems to be a direct consequence of the sync-subsumption. Thus, there is no hope for applying partial order reduction to the exploration of the local sync graph. So, if we are to develop a partial order reduction procedure based on the exploration of local zone graphs, we need to first develop a different subsumption relation that renders a finite local zone graph for which the independence relation between actions is easy to compute.

We observe that the critical difficulty in obtaining finite abstractions of local zone graphs lies in the arbitrary divergence between reference clocks of different processes. Keeping track of arbitrarily large differences between reference clocks is a major inherent difficulty when working with local zones. To avoid this problem, we will restrict our attention to those networks for which, given any feasible sequence of actions in its local zone graph, it is possible to find a run where the difference between reference clocks is bounded at all times. To make this idea precise, we introduce the concept of *spread* of a valuation, which is defined as the maximum difference between two reference clocks in the valuation. We say that a run is *D-spread-bounded* if the spread of all valuations in the run are bounded by a non-negative integer D. A network is *D-spread-bounded* if every local run has an equivalent run that is D-spread-bounded. If a network is D-spread-bounded for some D, we say the network is spread-bounded. Equipped with this notion, we restrict our attention to spread-bounded networks. A large number of models in

examples in the literature are in fact spread-bounded, and hence, we focus on designing efficient verification procedures for these systems.

We show that when computing a covering relation between two nodes of the local zone graph of a D-spread-bounded network, it is sufficient to consider containment with respect to the behaviour of valuations of spread D. We define a subsumption relation for local zone graphs that does precisely this. Thanks to this subsumption, we can obtain a finite transition system, for which we show that disjointness of domains allows us to compute independence of actions. This sets the stage for applying partial order reduction to the exploration of the local zone graphs of such networks.

We now give a brief overview of our new subsumption relation and the resultant transition system. We first propose a simulation relation for valuations in the local time semantics. Based on this simulation relation, we define an abstraction operator $\mathfrak{a}^*_M$ and a subsumption relation based on this abstraction that we refer to as $\mathfrak{a}^*_M$ *subsumption*. We show that $\mathfrak{a}^*_M$ subsumption is not finite in general. We consider a modified subsumption relation, which we refer to as $\mathfrak{a}^D_M$ subsumption, that is parameterized by a constant D. Rather than checking if a local zone Z is $\mathfrak{a}^*_M$-subsumed by another local zone Z', we check if $\mathsf{spread}_D(\mathsf{Z})$ is $\mathfrak{a}^*_M$-subsumed by $\mathsf{spread}_D(\mathsf{Z}')$, where $\mathsf{spread}_D(\mathsf{Z})$ is the set containing only the local valuations of spread D in Z. We denote this as $\mathsf{Z} \sqsubseteq^D_M \mathsf{Z}'$. This is enough as for a D-spread-bounded network, we need to focus only on valuations of spread D. We refer to the transition system obtained by applying $\mathfrak{a}^D_M$ subsumption to the local zone graph of a network as the $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}$ of the network. We prove that if a network is D-spread-bounded, then exploring the $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}$ of this network yields the correct answer for the reachability problem of the network. Essentially, the subsumption operation $\sqsubseteq^D_M$ is a generalization of the $\sqsubseteq^{\mathfrak{a}LU}_{\mathsf{sync}}$ operation (see Definition 5.5). While the $\sqsubseteq^{\mathfrak{a}LU}_{\mathsf{sync}}$ operator compares only the synchronized valuations in two zones, the $\sqsubseteq^D_M$ operator compares all valuations of spread at most D.

We show that actions with disjoint domains in the $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}$ of a D-spread-bounded network satisfies the diamond property (see conditions given in page 8). We remark that, just as in the case of LZG's, actions with disjoint domains do not satisfy the forward diamond property in $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}$'s.

We identify two classes of networks of timed automata, which we refer to as *global-local systems* and *client-server systems*. After giving some sufficient conditions for a network to have bounded spread, we propose two respective variants of the implementation - *global-local POR* and *client-server POR*. We remark that the global-local POR method is quite general in its applicability - it can be applied to any spread-bounded network, albeit with varying results depending on the nature of synchronizations. On the other hand, the client-server POR is focussed on networks where there is a centralized *server* process interacting with several *client* processes.

We also provide an evaluation of a prototype on some examples using the tool TChecker [HP19]. We first describe how our procedure works on some toy examples and then explain how we get gains for these systems. Further, we discuss the result of applying our procedure on some classical benchmarks for timed automata and discuss the performance on these models.

To summarize:

- We point out that we do not have an effective way to compute the independence relation between actions of a local sync graph. Consequently, we do not know how to apply partial order reduction directly on a local sync graph.
- This shows the need to develop a different subsumption relation for local zone graphs, one that renders a finite local zone graph that is amenable to partial order reduction. We define an alternate subsumption relation, denoted as $\mathfrak{a}^*_M$, that is based on a simulation relation between local valuations. However, $\mathfrak{a}^*_M$ subsumption is not finite in general.
- We introduce the notion of *spread* of a network of timed automata and give some sufficient conditions implying when a network is spread-bounded.
- We define a new subsumption relation, referred to as $\sqsubseteq^D_M$ subsumption whose application to the local zone graph of a network results in a finite transition system called $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}$ of the network. We prove that if a network is D-spread-bounded, then exploring the $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}$ of this network yields the correct answer for the reachability problem.
- We identify two classes of networks of timed automata, which we refer to as *global-local systems* and *client-server systems*, and propose specialized partial order reduction algorithms based on the exploration of $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}$, for spread-bounded networks belonging to these classes.
- We provide an evaluation of a prototype on some examples using the tool TChecker [HP19]. Our algorithm generates zone graphs that are an order of magnitude smaller than that produced by the standard reachability algorithm on some examples, while on some other examples, the size of the zone graph generated by our algorithm is larger than that of the standard zone graph. Here, we consider the number of nodes of the zone graph as a measure of the size of the zone graph.

1.5. Related work

In this section, we briefly overview work that is broadly focussed on alleviating the challenge posed by state-space explosion to the verification of concurrent

systems. We first discuss some of the landmarks in the development of partial order reduction techniques for networks of untimed systems. We then proceed to discuss some of the works that attempt to develop partial order reduction methods for concurrent timed systems.

State-space explosion

The challenge posed by state-space explosion to the verification of concurrent systems has been a subject of intensive research, and over the years, various solutions have been proposed to overcome this challenge (for an excellent survey, see [CKNZ11]). Techniques that use *symbolic representations* store sets of states and sets of transitions as opposed to explicitly representing each state/transition separately. Using symbolic representations can result in much smaller transition systems. For instance, in *symbolic model checking with binary decision diagrams (BDDs)* [CG18], a data structure called BDD is used to represent the sets of states and the transition relation of a transition system. This technique is observed to yield several orders of magnitude decrease in state space. Alternately, methods based on compositional reasoning [OG76, Lam77] analyse individual components of the network rather than the product transition system. *Counterexample-Guided Abstraction Refinement (CEGAR)* [CGJ$^+$03] uses the information obtained from the detection of counterexamples during the exploration to suitably tailor further exploration to a smaller state space. *Model Checking with SAT solvers* [BCC$^+$99] reduces the model checking problem to the satisfiability problem and uses the power of SAT-solvers (a widely studied class of tools) to answer the problem.

Partial order reduction is another such technique [Val89, Pel93, God90, AAJS17] and the development of foundations for partial order reductions for networks of timed automata is the central focus of this thesis. An alternate, but similar well-studied approach to combat state-space explosion is based on the idea of *unfoldings* [McM95, ERV02]. While the partial order reduction methods aim to explore only one representative from the set of all equivalent executions, the unfoldings based methods represent the set of all equivalent executions as a single partial order.

Partial order reduction for untimed systems

The study of developing partial order reduction methods for concurrent systems has been a topic of extensive research, and there is a large body of published studies devoted to the topic. Some of the widely used partial order reduction techniques are *Stubborn sets* by Valmari [Val90, Val89], *Ample sets* by Peled [Pel93, Pel96], and *Persistent sets* by Godefroid [God90, God96]. These techniques, which were developed independently, are similar in spirit but differ in the actual choice of the subsets that are computed. They have

been hugely successful in making the verification of concurrent systems more efficient.

The idea of *dynamic partial order reduction* was introduced by Flanagan et al. [FG05], where the independence relation between actions of the network is computed dynamically. This technique is observed to yield a greater reduction in state space than the static partial order reduction methods mentioned above. *Sleep sets* is an orthogonal approach introduced by Godefroid [God90] that focuses on reducing the number of transitions rather than the state space. Sleep sets can be used in conjunction with the ample/stubborn/persistent sets.

The development of partial order methods is still a very active area of research to this day, as evidenced by the recent development of *source sets* by Abdulla et al. [AAJS17]. In their work, the authors present source sets as a basis for optimal dynamic partial order reduction.

Local time semantics and issues of finiteness

The work that constitutes the primary basis for our work is by Bengtsson et al. [BJLY98], in which the authors introduce ideas of local time semantics and local zone graphs. While the local zone graph has several useful properties related to the independence of actions, it was not clear how to compute a finite local zone graph.

To this end, Minea [Min99b, Min99a] proposed a subsumption operation on local zones and designed a partial order reduction algorithm using this operation. Unfortunately, as we already mentioned, we demonstrate that this subsumption operation has a flaw that is not evident to repair. As a consequence, the partial order reduction procedure proposed in [Min99a] is not sound. As discussed in Section 1.4.1, we propose an alternate subsumption relation for local zone graphs, that proceeds via computation of aggregated zones.

Recall that coming to the same state with different zones inflicts a considerable blowup in the zone graph, since the same paths are explored independently from each of these zones. This has also been observed in the context of multi-threaded program verification by Sousa et al. [SRDK17]. Solving this problem in the context of program analysis requires over-approximation of the aggregated state. Fortunately, in the context of timed automata, the result of aggregating these zones is still a zone, as observed by Salah et al. [SBM06]. Our contribution towards efficient computation of aggregated zones is thus an important advance in timed automata verification.

Lugiez et al. [LNZ05] develop a partial-order semantics for networks of timed automata, which is similar in spirit to the local time semantics. However, the approach uses a different framework that is referred to as event zones. To obtain finiteness, an abstraction based on the maximal lower and upper bounds is employed. An exact correspondence between the work of

Lugiez et al. and our work on local zone graphs with sync-subsumption remains to be explored.

Partial order reduction for networks of timed automata

We have so far discussed works where the authors considered the local time semantics to obtain more commutativity between actions. Another approach to developing partial order reduction based procedures for timed systems involves detecting semi-commutations in the standard semantics. The idea of this approach can be summarized as follows: if a sequence of actions ab is guaranteed to produce a zone that is always contained in the zone produced by the sequence ba, then it is enough to explore only ba. Dams et al. [DGKK98] propose heuristics that statically find some pairs of actions like this. Since these are quite rare, Hansen et al. [HLL$^+$14], have developed a kind of CEGAR algorithm for dynamically finding such pairs. They have obtained significant gains on some examples.

An alternative approach to decrease the effect of state-space explosion for networks of timed automata, based on unfoldings introduced by Bouyer et al. [BHR06] and Cassez et al. [CCJ06, CJ06, CJ13]. In their works, the authors propose a verification algorithm that extends the unfolding based method to networks of timed automata. Here, they define a timed unfolding for a network of timed automata and propose an algorithm that computes a *finite prefix* of this timed unfolding. This finite prefix can be used to answer questions such as reachability for the network.

Partial order reduction procedures have also been proposed for other formalisms used to model real-time distributed systems, such as Time Petri Nets (TPN). Recently, in [BJL$^+$18], the authors have proposed an algorithm that applies stubborn set reduction to verification of timed-arc Petri nets with urgent behaviour.

Chapter 2

Preliminaries

In this chapter, we formally introduce the concepts, notations and definitions essential to understand this thesis. In the first part of this chapter, we introduce the notion of timed automata and discuss the standard approaches in the literature to solve the reachability problem for timed automata. In the latter part of this chapter, we introduce partial order reduction methods and discuss the basic principles of these techniques.

2.1. Timed automata

A clock is a variable that ranges over non-negative real numbers. We denote the set of clocks by X. Given a set of clocks X, let $\Phi(X)$ denote a set of clock constraints, where each clock constraint is a conjunction of comparisons of clocks with constants. Formally, $\Phi(X)$ is given by the grammar

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

where $x \in X$, $c \in \mathbb{N}$, and $\sim \in \{<, \leq, =, \geq, >\}$.

Definition 2.1 (Timed automaton [AD94])**.** A *timed automaton* A is given by a tuple $(Q, \Sigma, X, T, q_0, F)$, where Q is a finite set of states, X is the set of clocks, Σ is a finite alphabet of actions, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of accepting states and $T \subseteq Q \times \Sigma \times \Phi(X) \times 2^X \times Q$ is a transition relation. T contains transitions of the form (q, a, g, R, q'), where q is the source state, q' is the target state, a is the action triggering the transition, R is the set of clocks that are reset on the transition, and g is a *guard*, i.e., a clock constraint that needs to be satisfied for the transition to execute.

Remark. *We do not consider constraints of the form $x - y \sim c$ for guards in our timed automata. Constraints of the form $x - y \sim c$ are called* diagonal constraints*, and timed automata that allow diagonal constraints are typically harder to work with. It is known that diagonal constraints do not add any expressive power to timed automata and can be eliminated [AD94, BPDG98].*

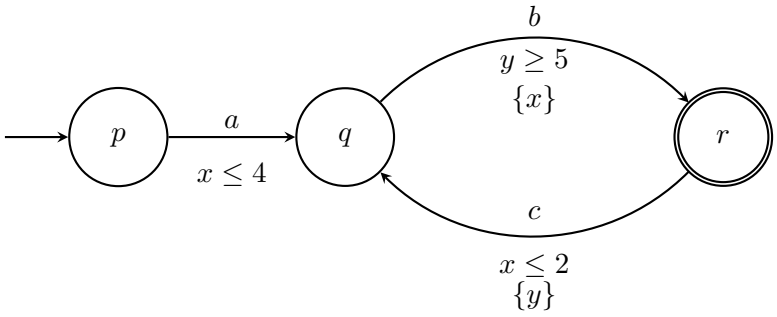


Figure 2.1: An example of a simple timed automaton A

Definition 2.2 (Standard valuation)**.** A standard valuation v is a function which maps every clock to a non-negative real value, i.e., $v : X \to \mathbb{R}_{\geq 0}$.

We now define some basic operations on valuations, namely, *guard satisfaction, reset*, and *time elapse*. We say that a valuation v satisfies a constraint ϕ, denoted $v \models \phi$, when each constraint in ϕ is satisfied when the variable x in the constraint is replaced by $v(x)$. Let R be a subset of the set of clocks X. We write $[R]v$ to denote the operation of resetting clocks in R; in other words, $[R]v$ is the valuation where clock x is 0 if $x \in R$, and it is $v(x)$ otherwise.

We denote by $v + \delta$ the valuation obtained by increasing the value of all clocks from the valuation v by $\delta \in \mathbb{R}_{\geq 0}$. It is the valuation obtained after a time elapse of δ time units from v. An example of a timed automaton is given in Figure 2.1. The timed automaton A has three states p, q and r and has two clocks, x and y. State p is the initial state, and r is the only accepting state of A. We can interpret the timing information associated with A as follows. The transition a from p to q has to be taken within 4 time units after the start of the run. Likewise, the transition b can only be taken 5 time units after the start. Note that the transition b resets the clock x. Since the next transition c has guard $x \leq 2$, c has to be taken within 2 time units of taking the transition b. The next b must occur at least 5 time units after c, and so on.

Definition 2.3. The semantics of a timed automaton A as in Definition 2.1, is given by an infinite-state transition system $\mathsf{TS}(A)$. The nodes of $\mathsf{TS}(A)$ are of the form (q, v), where q is a state of A and v is a standard valuation of the clocks of A. There are two kinds of transitions:

- delay transitions: $(q, v) \xrightarrow{\delta} (q, v + \delta)$,
- action transitions: $(q, v) \xrightarrow{a} (q', v')$, if $(q, a, g, R, q') \in T$, $v \models g$ and $v' = [R]v$.

The transition relation $\rightarrow$ is a union of all delay and action transitions. The initial state of $\mathsf{TS}(A)$ is the node $(q_0, \mathbf{0})$, where q_0 is the initial state and $\mathbf{0}$ is the valuation mapping all the clocks to 0. The accepting states are of the form (q, v), where $q \in F$ and v is an arbitrary valuation.

Definition 2.4 (Run of a timed automaton)**.** A *run* of a timed automaton is a sequence of transitions from the initial state of $\mathsf{TS}(A)$.

An example of a run of the timed automaton A from Figure 2.1 is

$$(p, x = 0, y = 0) \xrightarrow{\delta=3} \xrightarrow{a} (q, x = 3, y = 3) \xrightarrow{\delta=2} \xrightarrow{b} (r, x = 0, y = 5) \xrightarrow{\delta=2} \xrightarrow{c} (q, x = 2, y = 0) \cdots$$

A run is *accepting* if the final state of the run is accepting.

We are interested in the *reachability problem* of a timed automaton, which is defined formally as follows.

Definition 2.5 (Reachability problem)**.** The reachability problem for a timed automaton A is to decide if there is a run of A from an initial state to an accepting state.

As we already discussed in the introduction, in practice, a network of timed automata is often used rather than a single timed automaton. A network of timed automata is a set of timed automata operating concurrently.

Definition 2.6 (Network of timed automata)**.** Let $\Sigma_1, \ldots, \Sigma_k$ be finite alphabets, not necessarily mutually disjoint. Let $X_1, \ldots, X_k$ be mutually disjoint sets of clocks ($X_i \cap X_j = \emptyset$ for all pairs $i \neq j$). A network of timed automata is a tuple $\langle A_1, \ldots, A_k \rangle$, where each $A_i := (Q_i, \Sigma_i, X_i, T_i, q_0^i, F_i)$ is a timed automaton. The timed automata $A_1, \ldots, A_k$ are called *processes*. We write Proc to denote the set of all processes.

Figure 2.2 gives an example of a network of timed automata that consists of two processes A_1 and A_2.

Note that the set of actions Σ_i of a network are not necessarily disjoint. Given an action a, if $a \in \Sigma_i$, we say that A_i *participates* in a. We make this notion precise by defining the *domain* of an action.

Definition 2.7 (Domain of an action)**.** We define the domain of an action $a \in \Sigma$, denoted as $\mathsf{dom}(a)$, as the set of timed processes participating in the action a. Formally, $\mathsf{dom}(a) = \{i \mid a \in \Sigma_i\}$.

For instance, in the network of timed automata given in Figure 2.2, we have $\mathsf{dom}(a) = \{1\}$, $\mathsf{dom}(b) = \{2\}$ and $\mathsf{dom}(d) = \{1, 2\}$.

Definition 2.8. A transition whose domain contains more than one element is called a *synchronization transition*. A synchronization action could have guards and resets in each participating component. If the guard of a synchronization transition in a component of the network is satisfied, we say that the transition is *locally enabled* in that component.

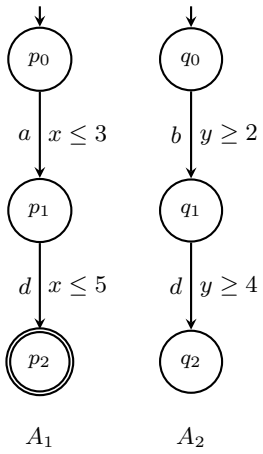


Figure 2.2: A network of timed automata

In the network given in Figure 2.2, observe that the action d is a synchronization transition. Given a standard valuation v, we see that d is locally enabled in A_1 if $v(x) \leq 5$, and locally enabled in A_2 if $v(y) \geq 4$.

Next, we define the semantics of a network of timed automata.

Definition 2.9 (Semantics of a network of timed automata)**.** The semantics of a network $\mathcal{N} := \langle A_1, \ldots, A_k \rangle$ is given by a timed automaton $A_{\mathcal{N}} = (Q_\times, \Sigma_\times, X_\times, T_\times, q^0_\times, F_\times)$, where

- $Q_\times = Q_1 \times Q_2 \times \cdots \times Q_k$
- $\Sigma_\times = \bigcup_{i=1}^{i=k} \Sigma_i$
- $X_\times = \bigcup_{i=1}^{i=k} X_i$
- $q^0_\times = (q^0_1, q^0_2, \ldots, q^0_k)$
- $F_\times = \{\ (q_1, q_2, \ldots, q_k) \mid q_i \in F_i \text{ for all } i\ \}$
- there is a transition of the form $(q_1, q_2, \ldots, q_k) \xrightarrow[R]{a,g} (q'_1, q'_2, \ldots, q'_k)$ in $T_\times$ when
  - (q_i, a, g_i, R_i, q'_i) in T_i for all $i \in \mathsf{dom}(a)$, $g := \bigwedge_i g_i$ and $R := \bigcup_i R_i$,
  - $q'_i = q_i$ for all $i \notin \mathsf{dom}(a)$,

The semantics of $\mathcal{N}$ is the transition system $\mathsf{TS}(A_{\mathcal{N}})$ of the automaton $A_{\mathcal{N}}$. We denote it $\mathsf{TS}(\mathcal{N})$ for short.

From the semantics of the network of timed automata, we can see that a synchronization transition can only be executed if it is locally enabled in all its participating components. Consider the network of timed automata

given in Figure 2.2. Here, d can only be executed from a standard valuation v that satisfies $v(x) \leq 5$ and $v(y) \geq 4$. Observe that all the components participating in a synchronization action are affected by (a possible) change of state, whereas those which do not participate remain in their current states.

2.2. Offset valuations

As discussed in the introduction, in this thesis, we consider two different semantics for networks of timed automata - the standard semantics (which we call global semantics) and the local time semantics. We would like to maintain a uniform representation for valuations in both semantics. To this end, rather than using the standard valuations, we choose to represent valuations of clocks using their *offsets* from the total time elapsed by the timed automaton. We discuss the offset representation of valuations (in the global semantics, and in later sections, the local time semantics) in detail. Since the offset representation of valuations in global semantics is non-standard, we present a translation between valuations in the standard representation and the offset representation.

For each clock $x \in X$, we introduce an offset clock variable $\widetilde{x}$, henceforth simply referred to as an *offset clock*. Let $\widetilde{X} = \{\widetilde{x} \mid x \in X\}$ be the set of all offset clocks. The value of $\widetilde{x}$ is the time-stamp at which x was last reset. We also introduce a variable t that tracks the global time: technically, t is a clock that is never reset. We refer to this clock as the *reference clock*. The idea is that the value of clock x is encoded as the difference $t - \widetilde{x}$. At the initial valuation, all the offset clocks and the reference clock have the same value. As time progresses, the offset clocks do not increase while the reference clock does, thereby increasing the difference between the reference clock and the offset clocks. Whenever a clock is reset, its offset clock is set to the value of the reference clock at the time of the reset.

Definition 2.10 (Offset valuations)**.** An offset valuation $\overline{v}$ is a function $\overline{v} : \widetilde{X} \cup \{t\} \mapsto \mathbb{R}_{\geq 0}$ such that $\overline{v}(\widetilde{x}) \leq \overline{v}(t)$ for all offset clocks $\widetilde{x} \in \widetilde{X}$.

Observe that in the offset representation, each variable $\widetilde{x} \in \widetilde{X}$ stores the time of last reset of x. Following this intuition, the time-stamp at which a clock is reset can never be greater than the total time elapsed, hence we have the constraint $\overline{v}(\widetilde{x}) \leq \overline{v}(t)$, for all offset valuations $\overline{v}$.

An *initial offset valuation* is any valuation in which all the offset clocks have the same value as the reference clock, i.e., $\overline{v}_0(t) = \overline{v}_0(\widetilde{x})$ for all offset clocks $\widetilde{x} \in \widetilde{X}$. Note that the actual value of the reference clock does not matter - what is important is that all the offset clocks and the reference clock have the same value.

Translations between offset and standard valuations

We introduce the operation std, which is a translation from offset valuations to standard valuations. Note that we use v to represent standard valuations, and $\overline{v}$ for offset valuations.

Definition 2.11 (std)**.** For a given offset valuation $\overline{v}$, we can obtain a standard valuation $\mathsf{std}(\overline{v})$:

$$\mathsf{std}(\overline{v})(x) = \overline{v}(t) - \overline{v}(\widetilde{x}), \qquad \text{for all } x \in X.$$

Conversely, given a standard valuation v, there are many offset valuations corresponding to it. One such offset valuation can be obtained by the following transformation:

$$\begin{aligned} \overline{v}(t) =& \max\{v(x) : x \in X\} \\ \overline{v}(\widetilde{z}) =& \overline{v}(t) - v(z) \qquad\qquad \text{for all } \widetilde{z} \in \widetilde{X}. \end{aligned}$$

Note that this is just one of the many offset valuations $\overline{v}$ such that $\mathsf{std}(\overline{v}) = v$. We illustrate this as follows: Let $\overline{v}$ be an offset valuation such that $\mathsf{std}(\overline{v}) = v$. Consider an offset valuation $\overline{v}'$ such that $\overline{v}'(t) = \overline{v}(t) + \delta$, and $\overline{v}'(\widetilde{x}) = \overline{v}(\widetilde{x}) + \delta$ for all offset clocks $\widetilde{x} \in \widetilde{X}$. It is easy to see that $\mathsf{std}(\overline{v}') = v$ too.

Operations on offset valuations

As we have seen, to give a semantics of a timed automaton we need some operations on clock valuations, namely, delay, reset, and guard satisfaction operations. In this section we discuss these operations for offset valuations.

- **Delay**: An offset variable $\widetilde{x}$ stores the time-stamp at which the clock x was last reset. Following this intuition, a delay operation increases the value of the reference clock t and leaves the offset clocks unchanged. Formally, for $\delta \in \mathbb{R}_{\geq 0}$, we denote by $\overline{v} + \delta$, the offset valuation defined by:

$$\begin{aligned} (\overline{v} + \delta)(t) &= \overline{v}(t) + \delta \\ (\overline{v} + \delta)(\widetilde{x}) &= \overline{v}(\widetilde{x}) \quad \text{for all} \quad \widetilde{x} \in \widetilde{X} \end{aligned}$$

- **Reset**: Resetting the clocks in $R \subseteq X$, yields an offset valuation $[R]\overline{v}$ obtained by setting the value of clocks that are reset to the value of the reference clock:

$$\begin{aligned} ([R]\overline{v})(t) &= \overline{v}(t) \\ ([R]\overline{v})(\widetilde{x}) &= \begin{cases} \overline{v}(t) & \text{if } x \in R, \\ \overline{v}(\widetilde{x}) & \text{otherwise} \end{cases} \end{aligned}$$

- **Guard satisfaction**: Given an offset valuation $\overline{v}$ and a clock constraint g, we say that $\overline{v}$ satisfies the guard g, written as $\overline{v} \models g$, if the standard valuation $\mathsf{std}(\overline{v})$ from Definition 2.11 satisfies g.

We now state Lemma 2.1 that relates the operations on offset valuations to the operations on standard valuations.

Lemma 2.1. *We have the following:*

- *If* $\overline{v} \models g$, *then* $\mathsf{std}(\overline{v}) \models g$.
- *If* $\overline{v}' = [R]\overline{v}$, *then* $\mathsf{std}(\overline{v}') = [R](\mathsf{std}(\overline{v}))$.
- *If* $\overline{v}' = \overline{v} + \delta$, *then* $\mathsf{std}(\overline{v}') = \mathsf{std}(\overline{v}) + \delta$ *for* $\delta \in \mathbb{R}_{\geq 0}$.

Proof. The first item follows from the definition of guard satisfaction for offset valuations.

For the second item, we know that $\overline{v}'(t) = \overline{v}(t)$, and $\overline{v}'(\widetilde{x}) = \overline{v}(\widetilde{x})$, if $x \notin R$ and $\overline{v}'(\widetilde{x}) = \overline{v}'(t)$ otherwise.

Let $v = \mathsf{std}(\overline{v})$ and $v' = \mathsf{std}(\overline{v}')$. Then, we observe that if $x \in R$, $v'(x) = \overline{v}'(t) - \overline{v}'(\widetilde{x}) = \overline{v}'(t) - \overline{v}'(t) = 0$. If $x \notin R$, $v'(x) = \overline{v}'(t) - \overline{v}'(\widetilde{x}) = \overline{v}(t) - \overline{v}(\widetilde{x}) = v(x)$. Thus, $v' = [R]v$.

For the third item, we know that $\overline{v}'(t) = \overline{v}(t) + \delta$, and $\overline{v}'(\widetilde{x}) = \overline{v}(\widetilde{x})$ for all $\widetilde{x} \in R$. Let $v = \mathsf{std}(\overline{v})$ and $v' = \mathsf{std}(\overline{v}')$. Then, it follows that $v'(x) = \overline{v}'(t) - \overline{v}'(\widetilde{x}) = \overline{v}(t) + \delta - \overline{v}(\widetilde{x}) = (\overline{v}(t) - \overline{v}(\widetilde{x})) + \delta = v(x) + \delta$. Thus, $v' = v + \delta$. □

Standard approaches to solve the reachability problem. We now discuss some standard approaches to solving the reachability problem of timed automata. One of the main challenges associated with studying problems pertaining to timed automata stems from the fact that the space of valuations of the timed automata is uncountably infinite. To work with timed automata we first need a way to handle this uncountable set of valuations effectively. There have been numerous attempts at doing this using different techniques [Bou09]. We discuss some of these approaches in the subsequent sections. The approaches we discuss employ the idea of using some symbolic representation to analyse these systems.

2.3. Regions

The earliest solution to deal with the uncountably infinite set of valuations was proposed by Alur et al. in [AD94]. Their approach involved partitioning the set of valuations into a finite number of sets, called *regions*. In this

section, we discuss their original approach and present a similar notion for the setting of offset valuations.

First, we present the definition of the classical region equivalence as stated in Alur et al. [AD94]. We proceed to propose a region equivalence for offset valuations, which leads to the notion of offset regions. We then prove that there is a bijection between the region equivalence in the standard setting (as introduced in [AD94]) and the region equivalence for offset valuations that we introduce here. Further, using this notion of regions, we introduce a transition system called the *offset region graph* of a timed automaton, which captures all the behaviours of the timed automaton. We show that the offset region graph gives us a way to solve the reachability problem, but this method turns out to be inefficient in practice.

For $x \in \mathbb{R}$, we write $\lfloor x \rfloor$ and $\{x\}$ to denote the integral part and fractional part of x, respectively. In other words, $\lfloor x \rfloor$ denotes the greatest integer smaller than or equal to x. For example, $\lfloor 7.3 \rfloor = 7$ and $\lfloor -2.1 \rfloor = -3$. The fractional part of x, written as $\{x\}$ is defined as $x - \lfloor x \rfloor$. Thus, $\{7.3\} = 0.3$ and $\lfloor -2.1 \rfloor = 0.9$. Note that $\{x\} \geq 0$ for all $x \in \mathbb{R}$.

Region equivalence for standard valuations

Definition 2.12 (Neighbourhood of a standard valuation ($\mathsf{nbd}(v)$))**.** A standard valuation v' is defined to be in the neighbourhood of v, written as $v' \sim v$, if for all clocks $x, y \in X$, we have:

- $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$,
- $\{v(x)\} = 0$ iff $\{v'(x)\} = 0$,
- $\{v(x)\} \leq \{v(y)\}$ iff $\{v'(x)\} \leq \{v'(y)\}$.

We denote by $\mathsf{nbd}(v)$ the set of all standard valuations in the neighbourhood of v.

Definition 2.13 (Classical region equivalence ($\sim_M$))**.** Let $M \in \mathbb{N} \cup \{-\infty\}$ be a constant. For a standard valuation v, we define $\mathsf{Bounded}_M(v) = \bigcup\{x \in X \mid v(x) \leq M\}$. Two standard valuations are said to be region equivalent w.r.t. M, written as $v \sim_M v'$ if

- $\mathsf{Bounded}_M(v) = \mathsf{Bounded}_M(v')$, and
- $v_{\restriction B} \sim v'_{\restriction B}$ where $v_{\restriction B}$ and $v'_{\restriction B}$ denote the standard valuations restricted to the variables in $B = \mathsf{Bounded}_M(v)$.

Given an automaton A, the bound M is obtained by choosing the maximum constant appearing in a guard of A. If there are no guards in A, then M is set to $-\infty$.

Observe that the notions of region and neighbourhood are closely related. If all clocks are bounded (which corresponds to having $M = \infty$) then the region of a standard valuation is the same as its neighbourhood.

Definition 2.14 (Region)**.** Let $M \in \mathbb{N} \cup \{-\infty\}$ be a constant. An M-region is an equivalence class of $\sim_M$. We simply say region instead of M-region, when M is clear from the context.

Figure 2.3 gives the regions for two clocks x and y with $M = 2$.

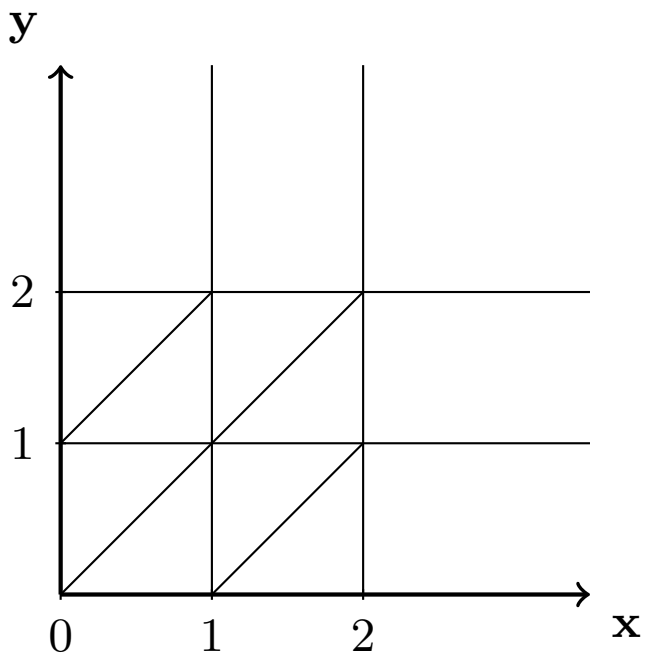


Figure 2.3: Regions of a timed automaton with two clocks x and y with $M = 2$. The set of regions consist of 9 corner points, 22 open line segments and 13 open regions [AD90].

Alternate representation of regions

Given a constant $M \in \mathbb{N} \cup \{-\infty\}$, each M-region can be uniquely identified by specifying the following information [AD94]:

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

$$\{x = c \mid c = 0, \cdots, M\} \cup \{c - 1 < x < c \mid c = 1, \cdots, M\} \cup \{x > M\}$$

2. for each pair of clocks $x, y \in X$ which satisfy the constraints of the form $c - 1 < x < c$ and $d - 1 < y < d$, whether $\{x\} < \{y\}$, $\{x\} = \{y\}$ or $\{x\} > \{y\}$.

Counting all the possible combinations of the constraints, it can be shown [AD94] that number of distinct M-regions is bounded by the function $|X|! \cdot 2^{|X|} \cdot \Pi_{x \in X}(2M + 2)$.

Region equivalence for offset valuations

We now introduce the notions of neighborhood and region equivalence in the context of offset valuations.

Definition 2.15 (Neighbourhood of an offset valuation)**.** For offset valuations $\overline{v}$ and $\overline{v}'$, we say $\overline{v}$ is in the neighbourhood of $\overline{v}'$, written as $\overline{v} \approx \overline{v}'$, if for all offset clocks $x, y \in \widetilde{X} \cup \{t\}$, we have $\lfloor \overline{v}(x) - \overline{v}(y) \rfloor = \lfloor \overline{v}'(x) - \overline{v}'(y) \rfloor$. The set of all offset valuations in the neighbourhood of $\overline{v}$ is denoted by $\mathsf{nbd}(\overline{v})$.

Definition 2.16 (Region equivalence for offset valuations ($\equiv_M$))**.** Let $M \in \mathbb{N} \cup \{-\infty\}$ be a constant. For an offset valuation $\overline{v}$, we define $\mathsf{Bounded}_M(\overline{v}) = \bigcup\{\widetilde{x} \in \widetilde{X} \mid \overline{v}(t) - \overline{v}(\widetilde{x}) \leq M\}$. We write $\overline{v} \equiv_M \overline{v}'$ if

- $\mathsf{Bounded}_M(\overline{v}) = \mathsf{Bounded}_M(\overline{v}')$
- $\overline{v}_{|B} \approx \overline{v}'_{|B}$ where $\overline{v}_{|B}$ and $\overline{v}'_{|B}$ denote the offset valuations restricted to the set of variables $B = \mathsf{Bounded}_M(\overline{v})$.

Definition 2.17 (Offset region)**.** Let $M \in \mathbb{N} \cup \{-\infty\}$ be a constant. An offset region is an equivalence class of $\equiv_M$. We write $[\overline{v}]^M$ for the offset region of $\overline{v}$.

We will now show that the notion of regions in the standard setting and offset setting are closely connected. But first, we prove a few technical lemmas which will be useful to prove this result.

Lemma 2.2. *For $x \in \mathbb{R}$, $\{x\} = 0$ iff $\lfloor x \rfloor = -\lfloor -x \rfloor$.*

Proof. $\{x\} = 0 \Leftrightarrow x = \lfloor x \rfloor$. Similarly, we have $\{x\} = 0 \Leftrightarrow -x = \lfloor -x \rfloor$. Since we know that $x = -(-x)$, we have $\{x\} = 0 \Leftrightarrow \lfloor x \rfloor = -\lfloor -x \rfloor$. □

Lemma 2.3. *For $x, y, z \in \mathbb{R}$, $\{z - x\} \leq \{z - y\}$ iff $\lfloor x - y \rfloor = \lfloor x - z \rfloor + \lfloor z - y \rfloor + 1$.*

Proof.

$$\begin{aligned} x - y &= x - z + z - y \\ &= \lfloor x - z \rfloor + \lfloor z - y \rfloor + \{x - z\} + \{z - y\} \\ &= \lfloor x - z \rfloor + \lfloor z - y \rfloor + 1 - \{z - x\} + \{z - y\} \end{aligned}$$

This gives the statement of the lemma. □

Lemma 2.4. *Let $\overline{v} \approx \overline{v}'$. Then, for the reference clock t and offset clocks $\widetilde{x}, \widetilde{y} \in \widetilde{X}$, we have $\{\overline{v}(t) - \overline{v}(\widetilde{x})\} \leq \{\overline{v}(t) - \overline{v}(\widetilde{y})\}$ iff $\{\overline{v}'(t) - \overline{v}'(\widetilde{x})\} \leq \{\overline{v}'(t) - \overline{v}'(\widetilde{y})\}$.*

Proof. Consider two offset clocks $\widetilde{x}$ and $\widetilde{y}$. From Lemma 2.3, we have the following conditions on $\overline{v}(\widetilde{x}) - \overline{v}(\widetilde{y})$ and $\overline{v}'(\widetilde{x}) - \overline{v}'(\widetilde{y})$:

$$\lfloor \overline{v}'(\widetilde{x}) - \overline{v}'(\widetilde{y}) \rfloor = \lfloor \overline{v}'(\widetilde{x}) - \overline{v}'(t) \rfloor + \lfloor \overline{v}'(t) - \overline{v}'(\widetilde{y}) \rfloor + 1 \text{ iff } \{\overline{v}'(t) - \overline{v}'(\widetilde{x})\} \leq \{\overline{v}'(t) - \overline{v}'(\widetilde{y})\}$$
$$\lfloor \overline{v}(\widetilde{x}) - \overline{v}(\widetilde{y}) \rfloor = \lfloor \overline{v}(\widetilde{x}) - \overline{v}(t) \rfloor + \lfloor \overline{v}(t) - \overline{v}(\widetilde{y}) \rfloor + 1 \text{ iff } \{\overline{v}(t) - \overline{v}(\widetilde{x})\} \leq \{\overline{v}(t) - \overline{v}(\widetilde{y})\}$$

Since $\overline{v} \approx \overline{v}'$, we know that

$$\lfloor \overline{v}(\widetilde{x}) - \overline{v}(\widetilde{y}) \rfloor = \lfloor \overline{v}'(\widetilde{x}) - \overline{v}'(\widetilde{y}) \rfloor$$
$$\lfloor \overline{v}'(\widetilde{x}) - \overline{v}'(t) \rfloor = \lfloor \overline{v}(\widetilde{x}) - \overline{v}(t) \rfloor$$
$$\lfloor \overline{v}'(t) - \overline{v}'(\widetilde{y}) \rfloor = \lfloor \overline{v}(t) - \overline{v}(\widetilde{y}) \rfloor$$

From the above conditions, it is easy to see that $\{\overline{v}'(t) - \overline{v}'(\widetilde{x})\} \leq \{\overline{v}'(t) - \overline{v}'(\widetilde{y})\}$ iff $\{\overline{v}(t) - \overline{v}(\widetilde{x})\} \leq \{\overline{v}(t) - \overline{v}(\widetilde{y})\}$. □

Lemma 2.5. *Suppose that $\overline{v} \approx \overline{v}'$. Then, for all offset clocks $\widetilde{x}, \widetilde{y} \in \widetilde{X}$, we have*

1. $\lfloor \overline{v}(t) - \overline{v}(\widetilde{x}) \rfloor = \lfloor \overline{v}'(t) - \overline{v}'(\widetilde{x}) \rfloor$
2. $\{\overline{v}(t) - \overline{v}(\widetilde{x})\} = 0$ *iff* $\{\overline{v}'(t) - \overline{v}'(\widetilde{x})\} = 0$
3. $\{\overline{v}(t) - \overline{v}(\widetilde{x})\} \leq \{\overline{v}(t) - \overline{v}(\widetilde{y})\}$ *iff* $\{\overline{v}'(t) - \overline{v}'(\widetilde{x})\} \leq \{\overline{v}'(t) - \overline{v}'(\widetilde{y})\}$

Proof. 1. Follows from the definition of $\approx$.

2. Let $\{\overline{v}(t) - \overline{v}(\widetilde{x})\} = 0$. Then, we know by Lemma 2.2 that $\lfloor \overline{v}(t) - \overline{v}(\widetilde{x}) \rfloor = -\lfloor \overline{v}(\widetilde{x}) - \overline{v}(t) \rfloor$. Since $\overline{v} \approx \overline{v}'$, we know that

$$\lfloor \overline{v}'(t) - \overline{v}'(\widetilde{x}) \rfloor = \lfloor \overline{v}(t) - \overline{v}(\widetilde{x}) \rfloor$$
$$\lfloor \overline{v}'(\widetilde{x}) - \overline{v}'(t) \rfloor = \lfloor \overline{v}(\widetilde{x}) - \overline{v}(t) \rfloor$$

   As a consequence, we have $\lfloor \overline{v}'(t) - \overline{v}'(\widetilde{x}) \rfloor = -\lfloor \overline{v}'(\widetilde{x}) - \overline{v}'(t) \rfloor$. By Lemma 2.2, this implies that $\{\overline{v}'(t) - \overline{v}'(\widetilde{x})\} = 0$.

3. Follows from Lemma 2.4. □

Lemma 2.6. *Suppose that $\overline{v} \approx \overline{v}'$. Then, $\mathsf{std}(\overline{v}) \sim \mathsf{std}(\overline{v}')$.*

Proof. Follows from Lemma 2.5 and the definition of the operation std. □

Lemma 2.7. *Suppose that $v \sim v'$. Then, $\overline{v} \approx \overline{v}'$, for arbitrary offset valuations $\overline{v}$ and $\overline{v}'$ such that $\mathsf{std}(\overline{v}) = v$ and $\mathsf{std}(\overline{v}') = v'$.*

Proof. Since $v \sim v'$, we know that $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$. This implies that for all $\widetilde{x} \in \widetilde{X}$, we have $\lfloor \overline{v}(t - \widetilde{x}) \rfloor = \lfloor \overline{v}'(t - \widetilde{x}) \rfloor$. From the definition of $\sim$ for standard valuations, we also know that $\{v(x)\} \leq \{v(y)\}$ if and only if $\{v'(x)\} \leq \{v'(y)\}$. This implies that $\{\overline{v}(t) - \overline{v}(\widetilde{x})\} \leq \{\overline{v}(t) - \overline{v}(\widetilde{y})\}$ if and only if $\{\overline{v}'(t) - \overline{v}'(\widetilde{x})\} \leq \{\overline{v}'(t) - \overline{v}'(\widetilde{y})\}$. But we know that

$$\overline{v}(\widetilde{x} - \widetilde{y}) = \lfloor \overline{v}(\widetilde{x} - t) \rfloor + \lfloor \overline{v}(t - \widetilde{y}) \rfloor + 1 - \{\overline{v}(t - \widetilde{x})\} + \{\overline{v}(t - \widetilde{y})\} \ .$$

Since $\lfloor \overline{v}'(t - \widetilde{y}) \rfloor = \lfloor \overline{v}(t - \widetilde{y}) \rfloor$ and $\lfloor \overline{v}'(\widetilde{x} - t) \rfloor = \lfloor \overline{v}(\widetilde{x} - t) \rfloor$ (as seen in the first part of the proof), using Lemma 2.3, it can be seen that $\lfloor \overline{v}'(\widetilde{x} - \widetilde{y}) \rfloor = \lfloor \overline{v}(\widetilde{x} - \widetilde{y}) \rfloor$. □

Lemma 2.8. $\overline{v} \approx \overline{v}'$ *iff* $\mathsf{std}(\overline{v}) \sim \mathsf{std}(\overline{v}')$.

Proof. The proof follows from Lemmas 2.6 and 2.7. □

Lemma 2.9. $\overline{v} \equiv_M \overline{v}'$ *iff* $\mathsf{std}(\overline{v}) \sim_M \mathsf{std}(\overline{v}')$.

Proof. Let $\overline{v} \equiv_M \overline{v}'$. From the definition of $\equiv_M$, we have $\mathsf{Bounded}(\overline{v}) = \mathsf{Bounded}(\overline{v}')$. This means that $\overline{v}(t) - \overline{v}(\widetilde{x}) > M$ if and only if $\overline{v}'(t) - \overline{v}'(\widetilde{x}) > M$ for an offset clock $\widetilde{x}$. As a consequence, $\mathsf{std}(\overline{v})(x) > M$ if and only if $\mathsf{std}(\overline{v}')(x) > M$. It immediately follows that $\mathsf{Bounded}(\mathsf{std}(\overline{v})) = \mathsf{Bounded}(\mathsf{std}(\overline{v}'))$. Further, since $\overline{v} \equiv_M \overline{v}'$, $\overline{v}_{\restriction B} \approx \overline{v}'_{\restriction B}$. From Lemma 2.6, we have $\mathsf{std}(\overline{v})_{\restriction B} \sim \mathsf{std}(\overline{v}')_{\restriction B}$.

In the converse direction, consider two standard valuations v and v' such that $v \sim_M v'$. Let $\overline{v}$ and $\overline{v}'$ be offset valuations such that $\mathsf{std}(\overline{v}) = v$ and $\mathsf{std}(\overline{v}') = v'$. We know that if $v(x) > M$, then $v'(x) > M$. By the definition of std operation, this implies that if $\overline{v}(t) - \overline{v}(\widetilde{x}) > M$, then $\overline{v}'(t) - \overline{v}'(\widetilde{x}) > M$. As a consequence, $\mathsf{Bounded}(\overline{v}) = \mathsf{Bounded}(\overline{v}')$. Moreover, from Lemma 2.7 and the fact that $v_{\restriction B} \sim v'_{\restriction B}$, we have $\overline{v}_{\restriction B} \approx \overline{v}'_{\restriction B}$. □

We will now state Lemma 2.10 that shows that given a maximal constant M, the number of offset regions is finite.

Lemma 2.10. *The number of regions as well as offset regions (Definition 2.17) is bounded by* $\mathcal{O}(|X|! \cdot 4^{|X|} \cdot (M+1)^{|X|})$.

Proof. By Lemma 2.9, we know that there is a bijection between the standard regions and offset regions.

Recall the alternate definition of standard regions given in Section 2.3. From this definition, we know that number of distinct regions is bounded by the function $|X|! \cdot 2^{|X|} \cdot \Pi_{x \in X}(2M + 2)$. Simplifying this expression, we get the bound $\mathcal{O}(|X|! \cdot 4^{|X|} \cdot (M+1)^{|X|})$. From the bijection between standard regions and offset regions given by Lemma 2.9, it follows that the number of offset regions is also bounded by the same function.

This implies the statement of the lemma. □

Time-abstract simulation

Since there are uncountably many valuations, we need some way to group valuations. Region equivalence is one such method. Here we consider a more general approach of grouping valuations that are not distinguishable with respect to the reachability of states.

We formalize this idea by defining a *time-abstract simulation* relation [TAKB96] between offset valuations.

Definition 2.18 (Time-abstract simulations)**.** Given a timed automaton $A := (Q, \Sigma, X, T, q_0, F)$, a *time-abstract simulation for* A is a relation $\preceq_A$ between the (state, offset valuation) pairs of A, such that, if $(q, \overline{v}) \preceq_A (q', \overline{v}')$, then

- $q' = q$;
- for every action a, delay $\delta \in \mathbb{R}_{\geq 0}$, and a transition of A of the form $(q, \overline{v}) \xrightarrow{\delta} \xrightarrow{a} (q_1, \overline{v}_1)$, there exists $\delta' \in \mathbb{R}_{\geq 0}$ such that $(q', \overline{v}') \xrightarrow{\delta'} \xrightarrow{a} (q'_1, \overline{v}'_1)$ and $(q_1, \overline{v}_1) \preceq_A (q'_1, \overline{v}'_1)$.

An offset valuation $\overline{v}$ is $\preceq_A$*-simulated by* $\overline{v}'$, in symbols $\overline{v} \preceq_A \overline{v}'$, if $(q, \overline{v}) \preceq_A (q, \overline{v}')$ for all states $q \in Q$ of A.

We will now show that the $\equiv_M$ relation between offset valuations is a time-abstract simulation relation for every timed automaton in which the maximum constant appearing in a guard is at most M.

Lemma 2.11. *Let* $\overline{v} \equiv_M \overline{v}'$*. For every delay* δ*, there exists a* δ' *such that* $\overline{v} + \delta \equiv_M \overline{v}' + \delta'$*.*

Proof. Consider differences w.r.t. offset clocks in $\overline{v} + \delta$ and $\overline{v}' + \delta'$. A delay of δ increases the value of differences of the form $t - \widetilde{x}$ by δ and decreases the differences of the form $\widetilde{x} - t$ by δ. Note that all the other differences of the form $\widetilde{x} - \widetilde{y}$ are not altered by a delay transition.

We fix $\lfloor \delta' \rfloor = \lfloor \delta \rfloor$. It can be observed that $\overline{v} + \lfloor \delta \rfloor \equiv_M \overline{v}' + \lfloor \delta' \rfloor$. Now, we need to fix $\{\delta'\}$ such that $\overline{v} + \delta \equiv_M \overline{v}' + \delta'$. Elapsing $\{\delta\}$ from $\overline{v} + \lfloor \delta \rfloor$ may either keep the difference $\overline{v}(t - \widetilde{x})$ in the same integer or move it up to the next integer, depending on the value of $\{\overline{v}(t - \widetilde{x})\}$.

Consider an ordering $\leq_{fr}$ of offset clocks such that $\widetilde{x} \leq_{fr} \widetilde{y}$ if $\{\overline{v}(t - \widetilde{x})\} \leq \{\overline{v}(t - \widetilde{y})\}$. Let the ordering $\leq_{fr}$ be of the form : $\widetilde{x}_1 \leq_{fr} \widetilde{x}_2 \leq_{fr} \cdots \leq_{fr} \widetilde{x}_k$. Let $\widetilde{x}_i$ be such that $\{\overline{v}(t - \widetilde{x}_i)\} + \{\delta\} < 1$ and $\{\overline{v}(t - \widetilde{x}_{i+1})\} + \{\delta\} \geq 1$. If no such $\widetilde{x}_i$ exists, it means that $\{\delta\}$ from $\overline{v} + \lfloor \delta \rfloor$ keeps all the differences $\overline{v}(t - \widetilde{x})$ in the same integer interval. In this case, we can choose $\delta' = \delta$. Otherwise, we choose $\{\delta'\}$ such that $1 - \{\overline{v}'(t - \widetilde{x}_i)\} > \{\delta'\} > 1 - \{\overline{v}'(t - \widetilde{x}_{i+1})\}$. By denseness of reals, such a δ' exists.

It remains to verify that $\overline{v} + \delta \equiv_M \overline{v}' + \delta'$. It is easy to see that if $(\overline{v} + \delta)(t - \widetilde{x}) > M$, then $(\overline{v}' + \delta')(t - \widetilde{x}) > M$. Hence, $\mathsf{Bounded}(\overline{v} + \delta) = \mathsf{Bounded}(\overline{v}' + \delta')$.

Next, consider the differences of the form $t - \widetilde{x}$ (where $\widetilde{x} \in \mathsf{Bounded}(\overline{v} + \delta)$). We have shown above that $\lfloor(\overline{v}'+\delta')(t-\widetilde{x})\rfloor = \lfloor(\overline{v}+\delta)(t-\widetilde{x})\rfloor$. By symmetry, we also have that $\lfloor(\overline{v}'+\delta')(\widetilde{x}-t)\rfloor = \lfloor(\overline{v}+\delta)(\widetilde{x}-t)\rfloor$. Lastly, consider differences of the form $\widetilde{x} - \widetilde{y}$ ($\widetilde{x}, \widetilde{y} \in \mathsf{Bounded}(\overline{v} + \delta)$.) Since the delay transition leaves these differences unaltered and integral values of such differences were equal in $\overline{v}'$ and $\overline{v}$, they continue to remain equal in $\overline{v}' + \delta'$ and $\overline{v} + \delta$. □

We define $(q, \overline{v}) \equiv_M (q, \overline{v}')$ when $\overline{v} \equiv_M \overline{v}'$.

Lemma 2.12. *$\equiv_M$ is a time-abstract simulation for A if the maximum constant used in a guard in A is at most M.*

Proof. Consider a delay transition of δ from $(q, \overline{v})$. Lemma 2.11 states that if $(q, \overline{v}) \xrightarrow{\delta} (q, \overline{v}_1)$ there is a delay $(q, \overline{v}') \xrightarrow{\delta'} (q, \overline{v}'_1)$ such that $\overline{v}_1 \equiv_M \overline{v}'_1$.

Next, consider an action transition a from $(q, \overline{v})$. Let a with the guard g reset the set of clocks R. Since $\overline{v} \equiv_M \overline{v}'$, $\overline{v}(t - \widetilde{x}) > M$ iff $\overline{v}'(t - \widetilde{x}) > M$ for all $\widetilde{x} \in \widetilde{X}$. Further, if $\widetilde{x} \in \mathsf{Bounded}(\overline{v})$, then $\lfloor\overline{v}(t - \widetilde{x})\rfloor = \lfloor\overline{v}'(t - \widetilde{x})\rfloor$ and $\lfloor\overline{v}(\widetilde{x} - t)\rfloor = \lfloor\overline{v}'(\widetilde{x} - t)\rfloor$. This implies that $\overline{v}$ satisfies g iff $\overline{v}'$ satisfies g. Therefore, $(q, \overline{v})$ can execute a iff $(q, \overline{v}')$ can execute a. Let $(q, \overline{v}) \xrightarrow{a} (q', \overline{v}_1)$ and $(q, \overline{v}') \xrightarrow{a} (q', \overline{v}'_1)$.

Next, we consider the reset operation. If x is reset, the differences $t - \widetilde{x}$ and $\widetilde{x} - t$ are set to 0 in the resultant offset valuation. This implies that for each clock $x \in R$, $t - \widetilde{x}$ and $\widetilde{x} - t$ are set to 0 in both $\overline{v}_1$ and $\overline{v}'_1$. Since resets leave the differences of the form $t - \widetilde{x}$ unchanged for $x \notin R$, the conditions w.r.t. these differences are satisfied. Next, consider the differences of the form $\widetilde{x} - \widetilde{y}$, where $x, y \in \mathsf{Bounded}(\overline{v}_1)$. If $x, y \in R$, we know that this difference is equal to 0 in both $\overline{v}_1$ and $\overline{v}'_1$. If $x \in R$ and $y \notin R$, we know that $\overline{v}_1(\widetilde{x} - \widetilde{y}) = \overline{v}(t - \widetilde{y})$. Similarly, we have $\overline{v}'_1(\widetilde{x} - \widetilde{y}) = \overline{v}'(t - \widetilde{y})$. Since $\overline{v} \equiv_M \overline{v}'$, we have $\lfloor\overline{v}_1(\widetilde{x} - \widetilde{y})\rfloor = \lfloor\overline{v}'_1(\widetilde{x} - \widetilde{y})\rfloor$ and $\lfloor\overline{v}_1(\widetilde{y} - \widetilde{x})\rfloor = \lfloor\overline{v}'_1(\widetilde{y} - \widetilde{x})\rfloor$. Finally, if $x, y \notin R$, we know that the difference $\widetilde{x} - \widetilde{y}$ are preserved from $\overline{v}$ to $\overline{v}_1$ (similarly for $\overline{v}'$ to $\overline{v}'_1$). Since $\overline{v} \equiv_M \overline{v}'$, the conditions w.r.t. differences of this form follows. Therefore, we have $\overline{v}_1 \equiv_M \overline{v}'_1$. □

2.4. Offset region graph

In Section 2.3, we introduced the notion of offset regions. We showed that offset valuations that belong to the same offset region are equivalent with respect to reachability. In other words, for two offset valuations $\overline{v}, \overline{v}'$ belonging to the same region, there is a run to a state q' from $(q, \overline{v})$ iff there is a run to q' from $(q, \overline{v}')$. Finally, we also showed that the number of distinct offset regions is bounded.

Equipped with the notion of offset regions, we now introduce a transition system that can be used to answer the reachability problem for timed

automata. We introduce the notion of *offset region graph*, a transition system whose nodes are (state, offset region) pairs.

Definition 2.19 (Offset region graph)**.** The offset region graph of a timed automaton A, denoted by $\mathsf{RG}(A)$, is a transition system whose nodes are of the form (q, R), where q is a state of A and R is an offset region w.r.t. the equivalence $\equiv_{M_A}$, where M_A is the maximum constant used in a guard in A. The initial node of the offset region graph is (q_0, R_0), where R_0 is the offset region containing all the initial valuations. A node (q, R) of the offset region graph is said to be accepting if q is an accepting state of A. $(q, R) \xrightarrow{t}_{\mathsf{RG}} (q', R')$ is a transition of the offset region graph if there are offset valuations $\overline{v}, \overline{v}'$ and a delay $\delta \in \mathbb{R}_{\geq 0}$ such that $\overline{v} \in R$, $\overline{v}' \in R'$ and $(q, \overline{v}) \xrightarrow{\delta} \xrightarrow{t} (q', \overline{v}')$.

Next, we show that transitions in the offset region graph faithfully represent transitions of the automaton. This feature of the offset region graph is known as the *pre-stability property* of offset regions.

Lemma 2.13. *Let $(q, R) \xrightarrow{t}_{\mathsf{RG}} (q', R')$ be a transition of the offset region graph. For every $\overline{v} \in R$, there is a delay $\delta' \in \mathbb{R}_{\geq 0}$ and $\overline{v}' \in R'$ such that $(q, \overline{v}) \xrightarrow{\delta} \xrightarrow{t} (q', \overline{v}')$.*

Proof. Since $(q, R) \xrightarrow{t}_{\mathsf{RG}} (q', R')$ is a transition of $\mathsf{RG}(A)$, there exist offset valuations $\overline{v} \in R, \overline{v}' \in R'$ and a delay $\delta \in \mathbb{R}_{\geq 0}$ such that $(q, \overline{v}) \xrightarrow{\delta} \xrightarrow{t} (q', \overline{v}')$.

Pick any offset valuation $\overline{v}_1$ from R. We know that $\overline{v}_1 \equiv_M \overline{v}$. By Lemma 2.12, we know that $(q, \overline{v}_1) \xrightarrow{\delta'} \xrightarrow{t} (q, \overline{v}'_1)$ such that $\overline{v}'_1 \equiv_M \overline{v}'$. Since $\overline{v}'_1 \equiv_M \overline{v}'$, we have $\overline{v}'_1 \in R'$. □

Lemma 2.14. *A has a run $(q, \overline{v}) \xrightarrow{\sigma} (q', \overline{v}')$ iff there is a path $(q, R) \xrightarrow{\sigma} (q', R')$ in $\mathsf{RG}(A)$ such that $\overline{v} \in R$ and $\overline{v}' \in R'$.*

Proof. Consider a run of A of the form

$$(q_0, \overline{v}_0) \xrightarrow{\delta_1} \xrightarrow{a_1} (q_1, \overline{v}_1) \xrightarrow{\delta_2} \xrightarrow{a_2} (q_2, \overline{v}_2) \cdots \xrightarrow{a_n} (q_n, \overline{v}_n).$$

By definition of $\mathsf{RG}(A)$, there exists a path in $\mathsf{RG}(A)$ of the form

$$(q_0, R_0) \xrightarrow{a_1} (q_1, R_1) \xrightarrow{a_2} (q_2, R_2) \cdots \xrightarrow{a_n} (q_n, R_n)$$

such that $\overline{v}_i \in R_i$ for $i \in \{0, 1, 2, \cdots, n\}$.

In the converse direction, consider a path of $\mathsf{RG}(A)$ of the form

$$(q_0, R_0) \xrightarrow{a_1} (q_1, R_1) \xrightarrow{a_2} (q_2, R_2) \cdots \xrightarrow{a_{n-1}} (q_{n-1}, R_{n-1}) \xrightarrow{a_n} (q_n, R_n)$$

We show here that there is a run of A of the form

$$(q_0, \overline{v}_0) \xrightarrow{\delta_1} \xrightarrow{a_1} (q_1, \overline{v}_1) \xrightarrow{\delta_2} \xrightarrow{a_2} (q_2, \overline{v}_2) \cdots \xrightarrow{a_{n-1}} (q_{n-1}, \overline{v}_{n-1}) \xrightarrow{a_n} (q_n, \overline{v}_n)$$

such that $\overline{v}_i \in R_i$ for all $i \in \{0, 1, 2, \cdots, n\}$. The proof follows by induction on the length of the path in $\mathsf{RG}(A)$. The base case is straightforward. For the induction step, consider the transition $(q_{n-1}, R_{n-1}) \xrightarrow{a_n} (q_n, R_n)$. By induction hypothesis, we know that there is a run

$$(q_0, \overline{v}_0) \xrightarrow{\delta_1} \xrightarrow{a_1} (q_1, \overline{v}_1) \xrightarrow{\delta_2} \xrightarrow{a_2} (q_2, \overline{v}_2) \cdots \xrightarrow{a_{n-1}} (q_{n-1}, \overline{v}_{n-1})$$

such that $\overline{v}_i \in R_i$ for $i \in \{0, 1, 2, \cdots, n-1\}$. In particular, we know that $\overline{v}_{n-1} \in R_{n-1}$. Using Lemma 2.13, we know that there is a delay $\delta_n \in \mathbb{R}_{\geq 0}$ such that $(q_{n-1}, \overline{v}_{n-1}) \xrightarrow{\delta_n} \xrightarrow{a_n} (q_n, \overline{v}_n)$ such that $\overline{v}_n \in R_n$. □

Corollary 2.1. *A has an accepting run iff there is a path in $\mathsf{RG}(A)$ from an initial node to an accepting node.*

The offset region graph gives us a way to solve the reachability problem of timed automata by exploring the offset region graph of the timed automaton. However, in practice, offset region graphs are too large to check for reachability in timed automata efficiently. In the next section, we discuss more efficient ways to solve the reachability problem.

2.5. Zones

The most widely used approach for checking reachability in a timed automaton is based on reachability in a graph called the *zone graph* of a timed automaton. In this section, we will discuss this approach in detail. We will introduce *zones* [BY03], which are sets of valuations that can be represented efficiently using difference constraints between clocks.

As discussed already, one of the main challenges encountered in the study of timed automata arises from the uncountably infinite space of its valuations. To overcome this problem, we use a symbolic representation to analyze these systems. Regions, that we introduced in Section 2.3, was one instance of such a symbolic representation of valuations. Building on the concept of regions, the idea of a region graph was introduced by Alur [AD94]. We presented a variant of the region graph in the offset setting, called offset region graph in Section 2.4. Intuitively, in the region graph of a timed automaton A, for each run in A, we have a path passing through all the regions of valuations in this run (the proof is similar to the proof of Lemma 2.14). However, from Lemma 2.10, we know that the number of regions grows exponentially with the number of clocks in the timed automaton. As a consequence, in practice, the region graph turns out to be too large to construct it efficiently. Zones are another way of grouping valuations. Although in principle, their number can also grow exponentially with the number of clocks, in most cases, the approach through zones is much more efficient than through regions.

Consider a sequence of actions of the timed automaton A. There are uncountably many runs of A with the same underlying action sequence - each one different from the others in the assignment to the clock variables in at least one step of the run. Since the timing information associated to these runs could be different, each run ends with a potentially different valuation. In other words, we end up with a set of valuations associated to this action sequence. For each sequence of actions, it turns out that a simple set of constraints can describe this associated set of valuations. In this section, we discuss this symbolic representation.

We first define a transition relation between sets of valuations. It will be very useful in the subsequent discussions.

Definition 2.20 (Symbolic transition)**.** Let A be a timed automaton and W be an arbitrary set of valuations of A. Given a transition t of A, we define $(q, W) \xrightarrow{t} (q', W')$ if $W' = \{v' \mid \exists v \in W, \exists \delta \in \mathbb{R}_{\geq 0} \text{ s.t. } (q, v) \xrightarrow{t} \xrightarrow{\delta} (q', v')\}$ is non-empty.

The symbolic transition computes the set W' by taking each valuation $v \in W$ that can execute the transition t, computing the valuation obtained after executing t from v and considering the set of all valuations that can reached by a time-elapse from the resulting valuation.

Next, we define some special sets of valuations, that can be defined using some simple constraints between the clocks. Let $X = \{x_1, x_2 \cdots, x_k\}$ be a set of clocks. We introduce a new clock x_0 that stands for the constant 0. We can then define an augmented set of clocks $X' = X \cup \{x_0\}$.

Definition 2.21 (Standard zones)**.** A standard zone is a set of valuations defined by a conjunction of constraints of the form $x - y \prec c$, where $x, y \in X'$ are clocks, $c \in \mathbb{Z}$ is a constant, and $\prec \in \{<, \leq\}$ stands for a strict, or non-strict inequality.

In the rest of this section, we will refer to standard zones simply as zones. A simple example of a zone over the set of clocks $\{x, y\}$ is:

$$(x_0 - x \leq 0) \wedge (x - x_0 \leq 2) \wedge (x_0 - y \leq -1) \wedge (x - y \leq 4) \ .$$

Note that another, shorter, representation of this zone is as follows

$$(x \geq 0) \wedge (x \leq 2) \wedge (y \geq 1) \wedge (x - y \leq 4) \ .$$

We will be using an analogous simplified representation for standard zones in the rest of this thesis. This zone is represented pictorially by the area shaded in red in Figure 2.4.

Bengtsson and Yi [BY03] show that if one starts with a set of valuations W that is a zone, then the set of valuations W' obtained on execution of a symbolic transition from W is also a zone. Further, let W_0 be the set of all

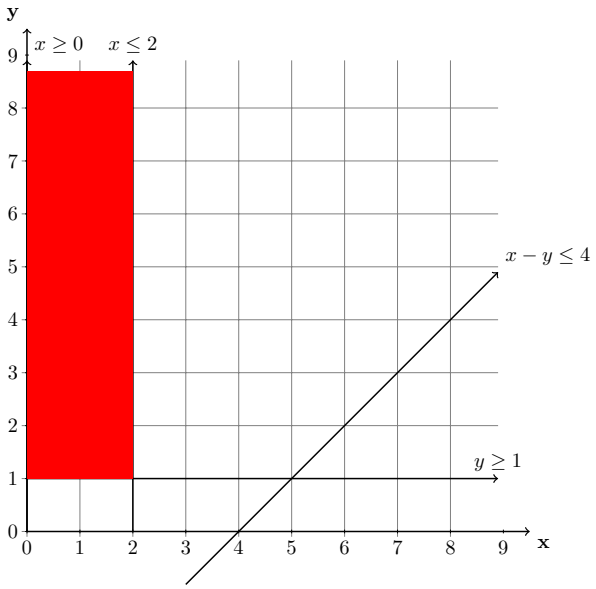


Figure 2.4: An example of a zone

valuations reachable by time-elapse from the initial valuation $\mathbf{0}$. The set of valuations W_0 can be described by a set of difference constraints between clocks, in the following way:

$$W_0 := \bigwedge_{x \in X} (x \geq 0) \wedge \bigwedge_{x,y \in X} (x - y = 0)$$

The constraint says that the values of all clocks are the same and at least 0. Clearly, W_0 is a zone.

Extending the above argument to a sequence of symbolic transitions, it follows that any set of valuations W that is reachable from (q_0, W_0) by a sequence of symbolic transitions is a zone. In other words, starting from (q_0, W_0), any sequence of symbolic transitions yields a node (q, W), where W is a zone. Using this idea, we build a transition system called the zone graph, which is discussed in detail in Section 2.7

Consider the zone given in Figure 2.4. Observe that the constraint $x - y \leq 4$ does not actually contribute to the definition of the zone. Even if we were to remove this constraint, or change this constraint to $x - y \leq 3$, the zone does not change. In such situations, we say that the constraint is *not tight* and that the zone is not *canonical*. We will now explain these concepts formally.

Before formalizing the notion of tightness of a constraint, we first define an order between the constraints used in the definitions of zones. Observe that in Definition 2.21, we consider constraints of the form $(\preccurlyeq, c)$ with $c \in \mathbb{Z}$. From now on, we also consider the constraint $(<, \infty)$, whose meaning will be explained at the time of its use.

Definition 2.22 (Ordering between constraints)**.** We have $(\prec_1, c_1) \Subset (\prec_2, c_2)$ when

- either $c_1 < c_2$
- or $c_1 = c_2$, and $\prec_1$ is $<$ while $\prec_2$ is $\leq$.

The intuition is that if we have $x - y \prec_1 c_1$ and $x - y \prec_2 c_2$, then the first constraint is more restrictive than the second.

Definition 2.23 (Tight constraints)**.** Let Φ be a set of constraints, and $\phi : x - y \prec_1 c_1$ be a constraint in Φ. We say that ϕ is tight in Φ if replacing ϕ in Φ by any another constraint $\phi' : x - y \prec_2 c_2$ such that $(\prec_2, c_2) \Subset (\prec_1, c_1)$ results in a different set of valuations. We say simply that ϕ is tight, when Φ is clear from the context.

Equipped with this definition, we can now formally define when a zone is canonical.

Definition 2.24 (Canonical form of a zone)**.** A zone Z is in canonical form if each constraint defining Z is tight.

Note that in the zone from Figure 2.4, if we change the constraint, $x - y \leq 4$ to $x - y \leq 2$, then the constraint is tight. This is because if we further tighten this constraint, then the set of valuations that satisfy this set of constraints is different from the original zone.

Representation of zones

Zones are commonly represented using *difference bound matrices* and *distance graphs*. In this section, we define these data structures and explain how they are used to represent zones.

Recall that X' is the original set of clocks X augmented with an additional clock x_0 representing the constant 0, and the ordering $\Subset$ between the constraints (see Definition 2.22). We now introduce some notation that will be useful in presenting both of these data structures.

Definition 2.25. For a zone Z and clocks $x_i, x_j \in X'$, we define $Z_{x_i x_j}$ as the $\Subset$-smallest constraint $(\prec, c)$ such that all valuations in Z satisfy $x_i - x_j \prec c$.

Observe that $Z_{x_i x_j}$ is always defined when Z is non-empty, since all valuations satisfy $x_i - x_j \prec \infty$.

Difference bound matrices (DBMs)[Dil89, Ben02]

A difference bound matrix (DBM) is a data structure that is widely used to represent zones. First introduced in [BM83], DBMs were reintroduced in the context of timed automata by Dill [Dil89].

Definition 2.26 (Difference bound matrix representation of a standard zone [Dil89])**.** Given a zone over a set of clocks $X = \{x_1, x_2, \cdots, x_n\}$, the difference bound matrix representation of Z is given by a $(n+1) \times (n+1)$ matrix M, where each entry of M is of the form $(\prec, c)$, where $c \in \mathbb{Z}$ and $\prec \in \{\leq, <\}$ or $(<, \infty)$. In particular, given a zone Z, the $(i,j)^{th}$ entry of the DBM of Z is $M_{ij} = (\prec, c)$, if there is a constraint $x_i - x_j \prec c$ in the set of constraints defining Z. If there is no constraint of the form $x_i - x_j \prec c$, then $M_{ij} = (<, \infty)$. If there are multiple constraints of the form $x_i - x_j \prec c$, we choose $(\prec, c)$ which is minimal w.r.t. the ordering $\Subset$ (see Definition 2.22.)

Given a difference bound matrix M, the zone defined by M is given by the set of valuations satisfying the conjunction of constraints of the form $x_i - x_j \prec_{ij} c_{ij}$, where $M_{ij} = (\prec_{ij}, c_{ij})$. In these constraints, the variable x_0 stands for the constant 0.

Distance graphs

We now discuss a graphical representation of zones, referred to as *distance graphs*. A distance graph is a representation of a zone as a weighted graph, whose vertex set is the augmented set of clocks $X' = X \cup \{x_0\}$, and the weight of the edge between a pair of vertices represents the constraint between the clocks that are its end-points. More precisely, between any two vertices, there is an edge with a weight $(\prec, c)$, where $c \in \mathbb{Z} \cup \{\infty\}$ and $\prec \in \{\leq, <\}$. An edge $x \xrightarrow{(\prec, c)} y$ in the distance graph is interpreted as the constraint $y - x \prec c$. Given a distance graph G, we use G_{xy} to denote the weight of the edge from x to y in G.

Definition 2.27 (Distance graph of a standard zone)**.** Given a zone Z over a set of clocks X, the distance graph of Z, denoted by G^Z, is given by the graph whose vertices are clocks from X' and edges $x \to y$ between each pair of vertices x, y of G_Z are such that the weight of the edge is $(\prec, c)$, if there is a constraint $y - x \prec c$ in the set of constraints defining Z. If there is no constraint of the form $y - x \prec c$, then the edge from x to y is labeled $(<, \infty)$ in G^Z. If there are multiple constraints of the form $y - x \prec c$, we choose the weight of the $x \to y$ edge to be $(\prec, c)$ that is minimal w.r.t. the ordering $\Subset$ (see Definition 2.22.)

Given a distance graph $G = (V, E)$, the zone defined by G, referred to as Z_G, is the set of valuations over the clocks $V(G)$, satisfying the conjunction of constraints of the form $y - x \prec c_{xy}$, for every edge in $E(G)$ of the form $x \xrightarrow{(\prec, c_{xy})} y$. For a distance graph G, we write $[\![G]\!]$ for the set of solutions to the constraints given by G.

The distance graph of the zone Z from Figure 2.4 is as depicted in Figure 2.5.

Observe that we can define the notion of a canonical distance graph, where the weight of each edge $x \to y$ in the distance graph G^Z is given

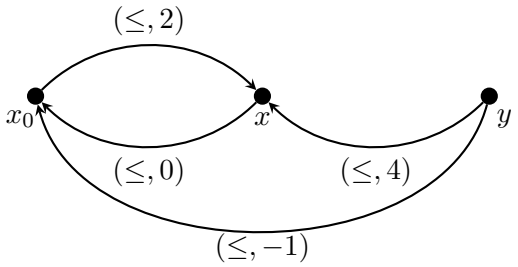


Figure 2.5: Distance graph of the zone given in Figure 2.4. All the edges that are not displayed have weight $(<, \infty)$.

by Z_{yx} is as defined in Definition 2.25. We now discuss how a zone can be converted into its canonical form. We present this using the distance graph representation of the zone. From the discussion on distance graphs and difference bound matrices, it is clear that one representation of a zone can be easily obtained from the other.

Canonicalization of a zone

Recall the definition of $\Subset$ ordering on constraints (Definition 2.22): $(\prec_1, c_1) \Subset (\prec_2, c_2)$. We define the operation of addition of constrains that permits us to formulate an important notion of negative cycle.

Definition 2.28 (Negative cycle in a distance graph)**.** We define an operation of addition on constraints:

$$\begin{aligned}(\prec_1, c_1) + (\prec_2, c_2) &= (<, \infty) && \text{if } c_1 = \infty \text{ or } c_2 = \infty \\ &= (\leq, c_1 + c_2) && \text{if } \prec_1 = \leq \text{ and } \prec_2 = \leq \\ &= (<, c_1 + c_2) && \text{otherwise}\end{aligned}$$

A cycle in a distance graph is *negative* if the sum of the edge weights along the cycle is strictly less in $\Subset$ ordering than $(\leq, 0)$.

Equipped with these definitions, we can now define when a distance graph is said to be in canonical form.

Definition 2.29. A distance graph G is said to be in canonical form if it has no negative cycle and for each pair of vertices x, y of G, the weight of the edge from x to y is not greater than the sum of the weights along any path from x to y.

For instance, consider the distance graph in Figure 2.5. The edge $y \to x$ has weight $(\leq, 4)$. However, the sum of the weights of the edges $y \to x_0$ and $x_0 \to x$ is $(\leq, 1)$, which is smaller than the $y \to x$ edge weight. Thus, the

distance graph is not in canonical form. The distance graph G in canonical form can be obtained by changing the edge weight $y \to x$ to $(\leq, 1)$.

Given a distance graph, we can compute the canonical form of the distance graph by running an all pairs shortest path algorithm, such as the Floyd-Warshall algorithm [BY03] on the distance graph. This algorithm has a running time of $\mathcal{O}(n^3)$, where n is the number of vertices in the graph. Since the distance graph of a zone over the set of clocks X has $|X| + 1$ vertices, the canonical form of zone can be computed in $\mathcal{O}(|X|^3)$ time.

Next, we state a folklore result [Sho81] connecting the emptiness of a zone with a property of its distance graph.

Proposition 2.1. *[Sho81] Let Z be a zone and let G_Z be its distance graph. Then, Z is empty if and only if G_Z has a negative cycle.*

This result implies that emptiness of a zone can be checked by looking for a negative cycle in its distance graph.

Intersection of distance graphs: Given two distance graphs G_1 and G_2 over the same set of vertices, the intersection of G_1 and G_2, denoted as $G_1 \cap G_2$, is given by the distance graph, where the weight of each edge is equal to the minimum w.r.t. $\Subset$ ordering, of the corresponding weights in G_1 and G_2. Note that, for any two clocks, $G_1 \cap G_2$ contains the tighter of the constraints between those in G_1 and those in G_2. So, each valuation in the zone defined by $G_1 \cap G_2$ satisfies the constraints from both G_1 and G_2. Thus, given two zones Z_1 and Z_2, the intersection of Z_1 and Z_2 is given by computing $G_1 \cap G_2$, where G_1 and G_2 are the distance graphs of Z_1 and Z_2, respectively.

2.6. Offset zones

In this section, we discuss *offset* zones, which are zones over valuations in the offset representation. We show their correspondence with zones over standard valuations defined in Section 2.5. In order to avoid confusion, we shall use $\mathcal{Z}$ to denote offset zones, while the standard zones will be denoted by Z.

Definition 2.30 (Offset zones)**.** An *offset zone* is a set of offset valuations determined by a conjunction of constraints of the form $\widetilde{x}_1 - \widetilde{x}_2 \prec\!\!\prec c$, where $\widetilde{x}_1, \widetilde{x}_2 \in \widetilde{X} \cup \{t\}$, $\prec\!\!\prec \in \{<, \leq\}$ and $c \in \mathbb{Z}$.

From the definition of an offset valuation, for any offset valuation $\overline{v}$, we have $\overline{v}(\widetilde{x}) \leq \overline{v}(t)$ for each offset variable $\widetilde{x} \in \widetilde{X}$. If each valuation in a zone $\mathcal{Z}$ satisfies a constraint ϕ, then we say that $\mathcal{Z}$ satisfies ϕ. Then, Lemma 2.15 follows.

Lemma 2.15. *An offset zone satisfies $\widetilde{x} \leq t$ for each $\widetilde{x} \in \widetilde{X}$.*

This implies that the set of constraints defining an offset zone will always include constraints of the form $\widetilde{x} \leq t$ for each offset clock $\widetilde{x} \in \widetilde{X}$.

An example of an offset zone over the set of clocks x, y is:

$$(\widetilde{x} - t \leq 0) \wedge (t - \widetilde{x} \leq 2) \wedge (\widetilde{y} - t \leq -1) \wedge (\widetilde{y} - \widetilde{x} \leq 4) \ .$$

Offset distance graphs

In this section, we discuss the distance graph representation for offset zones. The idea is quite similar to the distance graph representation of standard zones with only a few differences in details.

We introduce variables $\mathcal{Z}_{\widetilde{x}\widetilde{y}}$ for zone $\mathcal{Z}$ and offset clocks $\widetilde{x}, \widetilde{y} \in \widetilde{X} \cup \{t\}$. For defining $\mathcal{Z}_{\widetilde{x}\widetilde{y}}$, we use the $\Subset$ ordering between constraints as introduced in Definition 2.22.

Definition 2.31. For an offset zone $\mathcal{Z}$ and $x_i, x_j \in \widetilde{X} \cup \{t\}$, we define $\mathcal{Z}_{x_i x_j}$ as the $\Subset$-smallest constraint $(\prec, c)$ such that all valuations in $\mathcal{Z}$ satisfy $x_i - x_j \prec c$.

Note that $\mathcal{Z}_{x_i x_j}$ is always defined since all offset valuations satisfy $x_i - x_j \prec \infty$.

Definition 2.32 (Distance graph of offset zones)**.** Given a zone $\mathcal{Z}$ over a set of clocks $\widetilde{X}$, the distance graph of $\mathcal{Z}$, denoted by $G^{\mathcal{Z}}$, is given by the graph whose vertices are clocks from $\widetilde{X} \cup \{t\}$ and edges $\widetilde{x} \to \widetilde{y}$ between each pair of vertices $\widetilde{x}, \widetilde{y}$ of $G^{\mathcal{Z}}$ are such that the weight of the edge is $(\prec, c_{\widetilde{y}\widetilde{x}})$, if there is a constraint $\widetilde{y} - \widetilde{x} \prec c_{\widetilde{y}\widetilde{x}}$ in the set of constraints defining $\mathcal{Z}$. If there is no constraint of the form $\widetilde{y} - \widetilde{x} \prec c$, then the edge from $\widetilde{x}$ to $\widetilde{y}$ in $G^{\mathcal{Z}}$ is labeled $(<, \infty)$. If there are multiple constraints of the form $\widetilde{y} - \widetilde{x} \prec c$, we choose the weight of the $x \to y$ edge to be $(\prec, c)$ that is minimal w.r.t. the ordering $\Subset$ (see Definition 2.22.) We will write $G_{\widetilde{x}\widetilde{y}}$ to denote the weight of the edge $\widetilde{x} \to \widetilde{y}$ in the distance graph G.

Given a distance graph $G = (V, E)$, the zone defined by G, referred to as $\mathcal{Z}_G$, is the set of valuations over clocks $V(G)$, satisfying the conjunction of constraints of the form $\widetilde{x} - \widetilde{y} \prec c$, for every edge in $E(G)$ of the form $\widetilde{y} \xrightarrow{c_{\widetilde{x}\widetilde{y}}} \widetilde{x}$, where the weight of the edge $c_{\widetilde{x}\widetilde{y}}$ is $(\prec, c)$.

The distance graph of the offset zone from page 45 is depicted in Figure 2.6.

Remark (Comparison with a distance graphs of standard zones)**.** *Note that vertices of the distance graph of an offset zone $\mathcal{Z}$, are offset clocks $\widetilde{x}$ from $\widetilde{X}$ with an additional vertex for the reference clock t. This is because offset clocks*

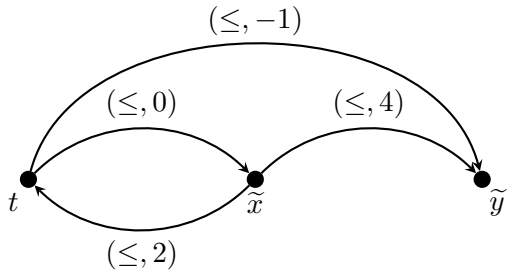


Figure 2.6: Distance graph of the offset zone given in page 45. All the edges that are not displayed have weight $(<, \infty)$.

are compared with t, in contrast with the standard setting, where clocks are compared with 0, represented by the clock x_0. This explains why in standard zones we have a vertex x_0, while in offset zones the role of x_0 is played by t.

Translation between offset zones and standard zones

We define an operation std which takes an offset zone and returns a standard zone.

Definition 2.33 (Standard zone from an offset zone)**.** Given an offset zone $\mathcal{Z}$, $\mathsf{std}(\mathcal{Z})$ is the zone given by the distance graph which is obtained by

- reversing the direction of all the edges in the distance graph of $\mathcal{Z}$,
- replacing the reference clock in the distance graph of $\mathcal{Z}$ by clock x_0, and
- replacing every offset clock $\widetilde{x}$ by the standard clock x.

We now provide an intuition for the transformation defined above. We use the relations between standard and offset zones described in Section 2.2. In particular we will use function std converting an offset valuation to a standard one. Let $\overline{v}$ be an offset valuation and $v = \mathsf{std}(\overline{v})$. Then, we know that

$$\begin{aligned} \overline{v}(\widetilde{x}) - \overline{v}(\widetilde{y}) = \ & \overline{v}(t) - \overline{v}(\widetilde{y}) - (\overline{v}(t) - \overline{v}(\widetilde{x})) \\ = \ & v(y) - v(x) \end{aligned}$$

Similarly,

$$\begin{aligned} \overline{v}(t) - \overline{v}(\widetilde{x}) = \ & v(x) - 0 \\ \overline{v}(\widetilde{x}) - \overline{v}(t) = \ & 0 - v(x) \end{aligned}$$

From the above observations, it is clear that if $\overline{v}$ satisfies a constraint $\widetilde{x} - \widetilde{y} \leq c$, then v satisfies the constraint $y - x \leq c$. Similarly, if $\overline{v}$ satisfies a constraint $\widetilde{x} - t \leq d$, then v satisfies the constraint $0 - x \leq d$.

Lemmas 2.16 and 2.17 below state that for every offset valuation $\overline{v}$ in $\mathcal{Z}$, $\mathsf{std}(\overline{v}) \in \mathsf{std}(\mathcal{Z})$, and for every standard valuation v in $\mathsf{std}(\mathcal{Z})$, if an offset valuation $\overline{v}$ satisfies $\mathsf{std}(\overline{v}) = v$, then $\overline{v}$ in $\mathcal{Z}$.

Lemma 2.16. *Let $\mathcal{Z}$ be an offset zone. For every valuation $\overline{v} \in \mathcal{Z}$, we have $\mathsf{std}(\overline{v}) \in \mathsf{std}(\mathcal{Z})$.*

Proof. Let $Z = \mathsf{std}(\mathcal{Z})$. Recall that in the distance graph of Z, G^Z, the weight of an edge is the weight of the reverse edge in the distance graph of $\mathcal{Z}$, $G^{\mathcal{Z}}$, with the modification that the reference clock t is replaced by the 0 clock and the offset clocks are replaced by the standard clocks.

Consider an offset valuation $\overline{v} \in \mathcal{Z}$. Let $v = \mathsf{std}(\overline{v})$. We know that $v(x) = \overline{v}(t) - \overline{v}(\widetilde{x})$ for all clocks $x \in X$. Hence, we have $v(x) - 0 = \overline{v}(t) - \overline{v}(\widetilde{x})$ for all clocks $x \in X$. Consequently, since $\overline{v}$ satisfies the constraints $\overline{v}(t) - \overline{v}(\widetilde{x}) \leq G^{\mathcal{Z}}_{\widetilde{x}t}$, we have $v(x) - 0 \leq G^Z_{0x}$ (since $G^{\mathcal{Z}}_{\widetilde{x}t} = G^Z_{0x}$) and since $\overline{v}$ satisfies $\overline{v}(\widetilde{x}) - \overline{v}(t) \leq G^{\mathcal{Z}}_{t\widetilde{x}}$, we have v satisfies $0 - v(x) \leq G^Z_{x0}$ (since $G^{\mathcal{Z}}_{t\widetilde{x}} = G^Z_{x0}$). Additionally, the value of differences between offset clocks also undergo a similar transformation on translation from the offset to the standard setting: i.e., $v(x) - v(y) = \overline{v}(\widetilde{y}) - \overline{v}(\widetilde{x})$ for all clocks $x, y \in X$. As a result, v satisfies all constraints of the form $v(x) - v(y) \leq G^Z_{yx}$, since $\overline{v}$ satisfies the constraint $\overline{v}(\widetilde{y}) - \overline{v}(\widetilde{x}) \leq G^{\mathcal{Z}}_{\widetilde{x}\widetilde{y}}$. This means that v satisfies all the constraints of Z and hence, we have $v \in Z$.

Therefore, for any offset valuation $\overline{v}$ in the offset zone $\mathcal{Z}$, the standard valuation $\mathsf{std}(\overline{v}) \in \mathsf{std}(\mathcal{Z})$. □

Lemma 2.17. *Let $\mathcal{Z}$ be an offset zone. For every $v \in \mathsf{std}(\mathcal{Z})$, and every offset valuation $\overline{v}$ such that $\mathsf{std}(\overline{v}) = v$, we have $\overline{v} \in \mathcal{Z}$.*

Proof. Let $Z = \mathsf{std}(\mathcal{Z})$ and $v \in Z$. Consider an arbitrary offset valuation $\overline{v}$, such that $\mathsf{std}(\overline{v}) = v$. We show below that $\overline{v} \in \mathcal{Z}$. Recall that in the distance graph of Z, G^Z, the weight of an edge is the weight of the reverse edge in the distance graph of $\mathcal{Z}$, $G^{\mathcal{Z}}$, with the reference clock t replaced by the 0 clock and the offset clocks replaced by the standard clocks.

We know that $v(x) = \overline{v}(t) - \overline{v}(\widetilde{x})$ for all clocks $x \in X$. This implies that, we have $v(x) - 0 = \overline{v}(t) - \overline{v}(\widetilde{x})$ for all $x \in X$. But we know that v satisfies the constraints $0 - v(x) \leq G^Z_{x0}$ and $v(x) - 0 \leq G^Z_{0x}$ for all clocks $x \in X$. Consequently, $\overline{v}$ will also satisfy the constraint $\overline{v}(\widetilde{x}) - \overline{v}(t) \leq G^{\mathcal{Z}}_{t\widetilde{x}}$ and $\overline{v}(t) - \overline{v}(\widetilde{x}) \leq G^{\mathcal{Z}}_{\widetilde{x}t}$, respectively, for all offset clocks $\widetilde{x} \in \widetilde{X}$. Additionally, the value of difference between clocks also undergo the following transformation on translation from the offset to the standard setting: i.e., $\overline{v}(\widetilde{x}) - \overline{v}(\widetilde{y}) = v(y) - v(x)$ for all clocks $x, y \in X$. Therefore, since v satisfies the constraint $v(y) - v(x) \leq G^Z_{xy}$, $\overline{v}$ will satisfy the constraint $\overline{v}(\widetilde{x}) - \overline{v}(\widetilde{y}) \leq G^{\mathcal{Z}}_{\widetilde{y}\widetilde{x}}$ for all offset clocks $\widetilde{x}, \widetilde{y} \in \widetilde{X}$.

This means that $\overline{v}$ satisfies all the constraints in $\mathcal{Z}$. Therefore, it follows that $\overline{v} \in \mathcal{Z}$. □

Operations on offset zones

Let g be a guard and R a set of clocks. We define the following operations on zones:

$$\begin{aligned} \mathcal{Z}_g &:= \{\overline{v} \in \mathcal{Z} \mid \overline{v} \models g\} \\ [R]\mathcal{Z} &:= \{[R]\overline{v} \mid \overline{v} \in \mathcal{Z}\} \\ \overrightarrow{\mathcal{Z}} &:= \{\overline{v} \mid \exists\, \overline{v}' \in \mathcal{Z},\ \exists \delta \in \mathbb{R}_{\geq 0} \text{ s. t. } \overline{v} = \overline{v}' + \delta\} \end{aligned}$$

We say that a zone is *time-elapsed* if $\mathcal{Z} = \overrightarrow{\mathcal{Z}}$.

Lemma 2.18. *Let $\mathcal{Z}$ be an offset zone. Then, $\mathcal{Z}_g$, $[R]\mathcal{Z}$ and $\overrightarrow{\mathcal{Z}}$ are offset zones.*

Proof. We will show that the sets of valuations obtained on applying these operations continue to be an offset zone. Let $G_{\mathcal{Z}}$ be the canonical distance graph of $\mathcal{Z}$. Note that we use the term *removing* an edge $x \to y$ from a distance graph to say that the value of this edge is assigned to $(<, \infty)$. We say a path is *lighter* than another to indicate that the weight of the former path is less than the weight of the latter path. In the same spirit, we use the term *lightest path* from x to y to refer to the path with the minimum weight from x to y.

Guard intersection: The set of constraints defining the offset zone $\mathcal{Z}_g$ is obtained by extending the set of constraints of $\mathcal{Z}$ with constraints $t - \widetilde{x} \sim c$, for each constraint of the form $x \sim c$ in g. This follows from the definition of when an offset valuation satisfies a guard, as given in Section 2.2.

Reset: Let G'_1 be the distance graph obtained by removing from $G_{\mathcal{Z}}$ all edges involving $\widetilde{x}$ and adding the edges $\widetilde{x} \xrightarrow{(\leq,0)} t$ and $t \xrightarrow{(\leq,0)} \widetilde{x}$, for each clock $x \in R$. Let G_1 be the distance graph obtained by canonicalizing G'_1. We will show that the set of valuations defined by G_1 is $[R]\mathcal{Z}$, i.e., $[\![G_1]\!] = [R]\mathcal{Z}$.

Consider edges of the form $\widetilde{y} \to \widetilde{z}$ in G_1, where $y, z \notin R$. We now show that the weight of such edges does not change from $G_{\mathcal{Z}}$ to G_1. First, observe that the first step in this transformation, i.e., removal of edges, cannot lead to lighter paths in distance graphs. Next, we inspect whether the newly added edges of weight $(\leq, 0)$ between $\widetilde{x}$ and t (from $\widetilde{x}$ to t and t to $\widetilde{x}$) where $x \in R$ can contribute to a lighter path from $\widetilde{y}$ to $\widetilde{z}$ in G'_1. Suppose that there was a lighter path from $\widetilde{y}$ to $\widetilde{z}$ using such a newly added edge between $\widetilde{x}$ and t. Consider the shortest and the lightest path.

- Suppose that the new lighter path used the $\widetilde{x} \xrightarrow{(\leq,0)} t$ edge. Since there are no other incoming edges to $\widetilde{x}$, the path cannot reach $\widetilde{x}$ without first going to t. However, this would imply that this lighter path is of the

form $\widetilde{y} \to \cdots \widetilde{u} \to t \xrightarrow{(\leq,0)} \widetilde{x} \xrightarrow{(\leq,0)} t \to \widetilde{s} \to \cdots \widetilde{z}$. Removing the part $t \xrightarrow{(\leq,0)} \widetilde{x} \xrightarrow{(\leq,0)} t$ of weight 0 yields a path $\widetilde{y} \to \cdots \widetilde{u} \to \widetilde{s} \to \cdots \widetilde{z}$ in $G_{\mathcal{Z}}$ whose weight is the same. But this is a contradiction, as we have taken the shortest path.

- Suppose that the new lighter path is of the form $\widetilde{y} \to \cdots t \to \widetilde{x} \to \widetilde{u} \to \cdots \to \widetilde{z}$. Recall that the only outgoing edge from $\widetilde{x}$ in G'_1 is to t. As a consequence, if there is an edge $\widetilde{x}$ to $\widetilde{u}$ in G_1, it is because of a path $\widetilde{x} \xrightarrow{(\leq,0)} t \to \widetilde{u}$. But this implies that our path can be rewritten as $\widetilde{y} \to \cdots t \xrightarrow{(\leq,0)} \widetilde{x} \xrightarrow{(\leq,0)} t \to \widetilde{u} \to \cdots \to \widetilde{z}$. The same argument as in the previous case applies.

  Once again, we can remove the smaller cycle $t \xrightarrow{(\leq,0)} \widetilde{x} \xrightarrow{(\leq,0)} t$ of weight 0 to obtain the path $\widetilde{y} \to \cdots \widetilde{u} \to \widetilde{s} \to \cdots \widetilde{z}$ in $G_{\mathcal{Z}}$, whose weight is lesser than the weight of the edge $\widetilde{y} \to \widetilde{z}$. As in the first case, this leads to a contradiction.

This shows that the weight of an edge $\widetilde{y} \to \widetilde{z}$ such that $y, z \notin R$ does not change from $G_{\mathcal{Z}}$ to G_1.
We will now show that $[\![G_1]\!] = [R]\mathcal{Z}$

$[R]\mathcal{Z} \subseteq [\![G_1]\!]$**:** Pick $\overline{v}' \in [R]\mathcal{Z}$. We know that $\overline{v}' = [R]\overline{v}$ for some $\overline{v} \in \mathcal{Z}$. We have $\overline{v}'(t) = \overline{v}(t)$, $\overline{v}'(t) - \overline{v}'(\widetilde{x}) = \overline{v}(t) - \overline{v}(\widetilde{x})$ if $x \notin R$ and $\overline{v}'(t) - \overline{v}'(\widetilde{x}) = 0$ if $x \in R$. It follows that $\overline{v}'$ satisfies all the constraints of G_1. Hence, $[R]\mathcal{Z} \subseteq [\![G_1]\!]$.

$[\![G_1]\!] \subseteq [R]\mathcal{Z}$**:** Consider a valuation $\overline{v} \in [\![G_1]\!]$. We construct a new distance graph G' whose edges are as follows:

- For clocks $x \notin R$, G' has the edges $t \xrightarrow{(\leq,\overline{v}(\widetilde{x})-\overline{v}(t))} \widetilde{x}$ and $\widetilde{x} \xrightarrow{(\leq,\overline{v}(t)-\overline{v}(\widetilde{x}))} t$.
- For clocks $x, y \notin R$, G' has the edge $\widetilde{x} \xrightarrow{(\leq,\overline{v}(\widetilde{y}-\widetilde{x}))} \widetilde{y}$.
- For clocks $x \in R$, there are no edges involving $\widetilde{x}$ in G'. This signifies that the weight of such an edge is $(<, \infty)$.

Notice that $\overline{v} \in [\![G']\!]$. Also, observe that G' is a canonical zone.

Observe that a solution to $G_{\mathcal{Z}} \cap G'$ gives a valuation $\overline{v}'$ such that

- $\overline{v}'$ satisfies all the constraints of $G_{\mathcal{Z}}$.
- $\overline{v}$ and $\overline{v}'$ agree on all the constraints involving clocks that are not reset.
- since $\overline{v} \in [\![G_1]\!]$, for all clocks $x \in R$, we have $\overline{v}(\widetilde{x}) = \overline{v}(t)$.

It is easy to see that $\overline{v}' \in \mathcal{Z}$ and $\overline{v}$ is obtained by a reset from $\overline{v}'$. Therefore, we have $\overline{v} \in [R]\mathcal{Z}$.

Now, we need to show that $G_{\mathcal{Z}} \cap G'$ is non-empty. Suppose that there is a negative cycle in $G_{\mathcal{Z}} \cap G'$. Since there were no negative cycles in $G_{\mathcal{Z}}$ or G', the new negative cycle should contain edges from both $G_{\mathcal{Z}}$ and G'. Further, since both $G_{\mathcal{Z}}$ and G' are canonical, the negative cycle should alternate between edges of $G_{\mathcal{Z}}$ and G'.

Consider an edge $\widetilde{x} \to \widetilde{y}$, where $x, y \notin R$. Recall that the weight of such edges does not change from $G_{\mathcal{Z}}$ to G_1. Since $\overline{v} \in [\![G_1]\!]$, we know that

$$\overline{v}(\widetilde{y}) - \overline{v}(\widetilde{x}) \leq c, \tag{2.1}$$

where $(\leq, c)$ is the weight of the edge $\widetilde{x} \to \widetilde{y}$ in $G_{\mathcal{Z}}$. Further, observe that the weight of the edge $\widetilde{x} \to \widetilde{y}$ in G' has weight $\overline{v}(\widetilde{y}) - \overline{v}(\widetilde{x})$. From 2.1, we can see that this edge in $G_{\mathcal{Z}} \cap G'$ has to be from G'. Thus, all edges in $G_{\mathcal{Z}} \cap G'$ of the form $\widetilde{x} \to \widetilde{y}$ where $x, y \notin R$ come from G'. This implies that the negative cycle cannot be limited to the offset clocks that were not reset.

Also, observe that since there are no edges in G' involving clocks $\widetilde{x}$ where $x \in R$, any edges involving these clocks must come from $G_{\mathcal{Z}}$. Suppose our negative cycle contained a clock $\widetilde{x}$ such that $x \in R$. By our criterion for the negative cycle, either the incoming or the outgoing edge associated to $\widetilde{x}$ should come from G', which we know is not possible as there are no edges associated to s in G'. Consequently, this implies that the negative cycle cannot involve clocks that are reset.

Thus, from the above observations, it is clear that the negative cycle cannot be restricted to just offset clocks - it should pass through t.

Let the negative cycle be of the form

$$\widetilde{x} \to \cdots \to \widetilde{y} \to t \to \widetilde{z} \to \widetilde{u} \to \cdots \to \widetilde{x}$$

Suppose that the edge $t \to \widetilde{z}$ comes from G'. Then, the edge $\widetilde{z} \to \widetilde{u}$ should come from $G_{\mathcal{Z}}$. However, since $u \notin R$, we know that the edge $\widetilde{z} \to \widetilde{u}$ in $G_{\mathcal{Z}} \cap G'$ comes from G', which is contrary to our requirement. Thus, the edge $t \to \widetilde{z}$ comes from $G_{\mathcal{Z}}$. As a consequence, the incoming edge to t, namely, $\widetilde{y} \to t$, should be from G'.

We have already seen that all the edges of the form $\widetilde{x} \to \widetilde{y}$ such that $x, y \notin R$ can only be from G'. Since G' is canonical we can replace all these edges by a single edge to get the negative cycle

$$\widetilde{x} \to t \to \widetilde{z} \to \widetilde{x}$$

Further, since $\widetilde{x} \to t$ and $\widetilde{z} \to \widetilde{x}$ are both from G', we can replace it by a single edge. Thus, the situation boils down to a negative cycle of just 2 edges of the form

$$\widetilde{z} \xrightarrow{(\lessdot, k_1)} t \xrightarrow{(\lessdot, k_2)} \widetilde{z}$$

where the edge $\widetilde{z} \xrightarrow{(\prec_1, k_1)} t$ is from G' and the edge $t \xrightarrow{(\prec_2, k_2)} \widetilde{z}$ is from $G_{\mathcal{Z}}$.

From our assumption, since this is a negative cycle, we have $(\prec_1, k_1) + (\prec_2, k_2) < (\leq, 0)$. Since we know that $k_1 = \overline{v}(t) - \overline{v}(\widetilde{z})$, this implies

$$\overline{v}(t) - \overline{v}(\widetilde{z}) + k_2 < 0$$

We know that the edge $t \to \widetilde{z}$ in G_1 is unchanged from $G_{\mathcal{Z}}$, since $z \notin R$. Further, since we know that $\overline{v} \in [\![G_1]\!]$, we have $\overline{v}(\widetilde{z}) - \overline{v}(t) \leq k_2$, which implies

$$\overline{v}(t) - \overline{v}(\widetilde{z}) + k_2 \geq 0$$

This is a contradiction. Thus, our assumption that there was a negative cycle was wrong.

Time-elapse: Let G_1 be the distance graph obtained by removing from $G_{\mathcal{Z}}$, the edges of the form $\widetilde{x} \to t$ for each $x \in X$. We will show that $[\![G_1]\!] = \overrightarrow{\mathcal{Z}}$.

First observe that G_1 is canonical. We know that $G_{\mathcal{Z}}$ is canonical - so, the lightest path between any two clocks $\widetilde{x}$ and $\widetilde{y}$ is the weight of the edge between the two clocks. The transformation from $G_{\mathcal{Z}}$ to G_1 only involves the removal of some edges which cannot lead to lighter paths.

$\overrightarrow{\mathcal{Z}} \subseteq [\![G_1]\!]$: Let $\overline{v}' \in \overrightarrow{\mathcal{Z}}$. We know that $\overline{v}' = \overline{v} + \delta$ for some $\overline{v} \in \mathcal{Z}$. This means that $\overline{v}'$ is obtained by increasing the value of the reference clock t from $\overline{v}$, while keeping the values of the offset clocks unchanged. This gives $\overline{v}'(\widetilde{x}) - \overline{v}'(\widetilde{y}) = \overline{v}(\widetilde{x}) - \overline{v}(\widetilde{y})$ and $\overline{v}'(t) - \overline{v}'(\widetilde{x}) \geq \overline{v}(t) - \overline{v}(\widetilde{x})$ for all offset clocks $\widetilde{x}, \widetilde{y}$. Since $\overline{v} \in \mathcal{Z}$ and hence satisfies the constraints in $[\![G_1]\!]$, it follows that $\overline{v}'$ satisfies them as well.

$[\![G_1]\!] \subseteq \overrightarrow{\mathcal{Z}}$: Consider $\overline{v} \in [\![G_1]\!]$.

We construct a new distance graph G' with the following edge information:

- For all clocks $\widetilde{x} \in \widetilde{X}$, we have the edges $\widetilde{x} \xrightarrow{(\leq, \overline{v}(t) - \overline{v}(\widetilde{x}))} t$.
- We do not have edges of the form $t \to \widetilde{x}$.
- For clocks $\widetilde{x}, \widetilde{y} \in \widetilde{X}$, we have edges $\widetilde{x} \xrightarrow{(\leq, \overline{v}(\widetilde{y}) - \overline{v}(\widetilde{x}))} \widetilde{y}$.

Observe that G' is canonical. We remark that a valuation that satisfies the constraints in G' need not satisfy $\widetilde{x} \leq t$, and consequently, need not be an offset valuation. Further, note that $[\![G']\!]$ is non-empty as $\overline{v} \in [\![G']\!]$. Thus, G' does not have a negative cycle.

A solution to $G_{\mathcal{Z}} \cap G'$ gives a valuation $\overline{v}'$ such that $\overline{v}' \in \mathcal{Z}$ and $\overline{v}$ is obtained by a time elapse elapse from $\overline{v}'$, and hence will imply $\overline{v} \in \overrightarrow{\mathcal{Z}}$. We will now show that $G_{\mathcal{Z}} \cap G'$ is non-empty. Suppose that there is a negative cycle in $G_{\mathcal{Z}} \cap G'$. Since there was no negative cycle in $G_{\mathcal{Z}}$ or G', the new negative cycle should contain edges from both the distance graphs. Further,

since both $G_{\mathcal{Z}}$ and G' are canonical, the negative cycle should alternate between edges of $G_{\mathcal{Z}}$ and G'.

Recall that the edges between offset clocks, say $\widetilde{x}$ and $\widetilde{y}$, do not change from $G_{\mathcal{Z}}$ to G_1. Since $\overline{v} \in [\![G_1]\!]$, it follows that $\overline{v}(\widetilde{x}) - \overline{v}(\widetilde{y}) \leq c$, where $(\leq, c)$ is the weight of the edge $\widetilde{x} \to \widetilde{y}$ in $G_{\mathcal{Z}}$. In other words, for each edge of the form $\widetilde{x} \to \widetilde{y}$, the weight of this edge in G' is smaller than the weight of the edge in $G_{\mathcal{Z}}$. As a consequence, we know that all edges of the form $\widetilde{x} \to \widetilde{y}$ in $G_{\mathcal{Z}} \cap G'$ come from G'.

From the above observations, it is clear that the negative cycle should pass through t. Let the negative cycle be of the form $\widetilde{z} \to \cdots \widetilde{x} \to t \to \widetilde{y} \to \cdots \to \widetilde{z}$. Since the edge $t \to \widetilde{y}$ has weight $(<, \infty)$ in G', we know that this edge should come from $G_{\mathcal{Z}}$. Further, since the edges should alternate between $G_{\mathcal{Z}}$ and G', the $\widetilde{x} \to t$ edge should come from G'.

Consider the segment $\widetilde{x} \to t \to \widetilde{y}$ of this negative cycle - let this segment have weight $(\lessdot, k)$. Clearly, $(\lessdot, k)$ is less than the weight of the edge $\widetilde{x} \to \widetilde{y}$ in G' (which is equal to $(\leq, \overline{v}(\widetilde{y} - \widetilde{x}))$), or we would have used this edge for the negative cycle. Thus, $(\leq, \overline{v}(\widetilde{y} - \widetilde{x})) > (\lessdot, k)$ which implies $(\leq, \overline{v}(\widetilde{x} - \widetilde{y})) + (\lessdot, k) < (\leq, 0)$. This implies that $\widetilde{x} \to t \to \widetilde{y} \to \widetilde{x}$ is a negative cycle. Since both $\widetilde{x} \to t$ and $\widetilde{y} \to \widetilde{x}$ edges come from G' and G' is canonical, we can replace these two edges by a single edge. We now have the negative cycle $\widetilde{y} \to t \to \widetilde{y}$, where $\widetilde{y} \to t$ edge comes from G' with weight $(\leq, \overline{v}(t - \widetilde{y}))$, and $t \to \widetilde{y}$ edge with weight $(\lessdot, k)$ comes from $G_{\mathcal{Z}}$.

Recall that $\overline{v} \in [\![G_1]\!]$, and G_1 has the same incoming edges to t as in $G_{\mathcal{Z}}$. This implies that $\overline{v}(\widetilde{y} - t) \lessdot k$. In other words, we have

$$(\leq, \overline{v}(\widetilde{y} - t)) \leq (\lessdot, k) \tag{2.2}$$

We know that $\widetilde{y} \xrightarrow{(\leq, \overline{v}(t-\widetilde{y}))} t \xrightarrow{(\lessdot, k)} \widetilde{y}$ is a negative cycle. From 2.2, we know that if we replace the weight of the $t \to \widetilde{y}$ edge by $(\leq, \overline{v}(\widetilde{y} - t))$, which is the weight of the $t \to \widetilde{y}$ edge in G', we should still have a negative cycle. But this is a contradiction, as we know that G' does not have negative cycles. Thus, our assumption that $G_{\mathcal{Z}} \cap G'$ has a negative cycle was wrong. As a consequence, we know that $\mathcal{Z} \cap [\![G']\!]$ is not empty. □

2.7. Offset zone graph

As already discussed in Section 2.5, if we start with (q, W), where W is a standard zone, then the set of valuations W' in (q', W') obtained after an execution of a symbolic transition (see Definition 2.20) is also a standard zone [BY03]. Further, the set of all valuations reachable by time-elapse from the initial valuation v_0 is also a zone. It follows that if we start from (q_0, W_0), any sequence of symbolic transitions yields a node (q, W), where W is a zone.

This observation allows to construct a transition system called the *zone graph*, whose nodes are (state, zone) pairs [BY03]. We have already discussed offset zones and have established the correspondence between standard zones and offset zones in Section 2.6. In this section, we build on these ideas and propose the notion of a transition system called *offset zone graph*, whose nodes are (state, offset zone) pairs. This graph may be viewed as the analogue of the standard zone graph in the offset setting.

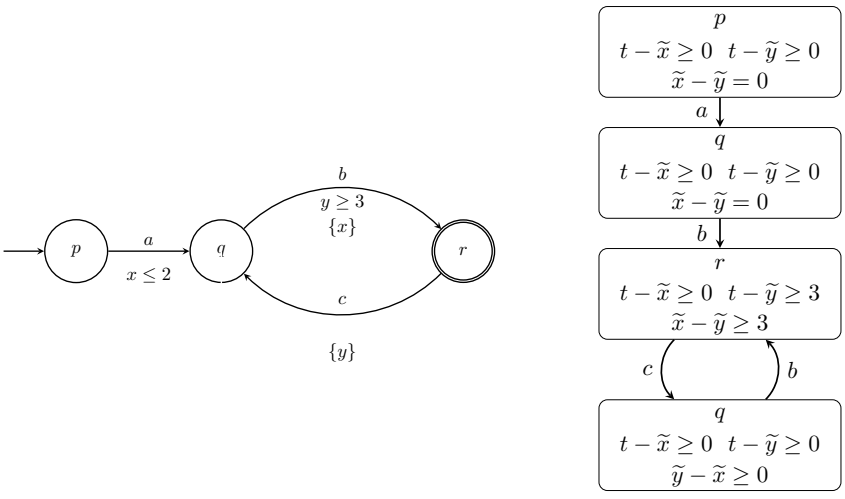
Definition 2.34 (Offset zone graph)**.** The offset zone graph $\mathsf{OZG}(A)$ of a timed automaton $A := (Q, \Sigma, X, T, q_0, F)$ is a transition system whose

- nodes are of the form $(q, \mathcal{Z})$, where $q \in Q$ and $\mathcal{Z}$ is a time-elapsed offset zone.
- transition relation is defined as follows:

$$(q, \mathcal{Z}) \xrightarrow{a} (q', \mathcal{Z}') \text{ if}$$

  $q \xrightarrow[R]{a,g} q'$ is a transition in T and $\mathcal{Z}' = \overrightarrow{[R](\mathcal{Z} \wedge g)}$ is non-empty.
- initial state is of the form $(q_0, \mathcal{Z}_0)$, where $\mathcal{Z}_0$ is the initial zone, which is the set of valuations obtained by time-elapse from an initial valuation $\overline{v}_0$, i.e., $\mathcal{Z}_0 = \{\overline{v}_0 + \delta \mid \delta \in \mathbb{R}_{\geq 0}\}$.
- accepting state is of the form $(q, \mathcal{Z})$, where $q \in F$.

Figure 2.7 gives a timed automaton and its offset zone graph.



(a) A timed automaton A (b) Offset zone graph of A

Figure 2.7: A timed automaton and its offset zone graph

We state below the *Pre and Post properties* of runs on offset zones.

Lemma 2.19 (Pre and Post properties of offset zones)**.** *Let A be a timed automaton and $\mathsf{OZG}(A)$ be the offset zone graph of A. Let σ be a sequence of actions.*

- *Suppose that $(q, \overline{v}) \xrightarrow{\sigma} (q', \overline{v}')$ is a run of A and $(q, \mathcal{Z})$ is a node of $\mathsf{OZG}(A)$ with $\overline{v} \in \mathcal{Z}$. Then, there is a path $(q, \mathcal{Z}) \xrightarrow{\sigma} (q', \mathcal{Z}')$ in $\mathsf{OZG}(A)$ such that $\overline{v}' \in \mathcal{Z}'$.*
- *Suppose that $(q, \mathcal{Z}) \xrightarrow{\sigma} (q', \mathcal{Z}')$ is a path in $\mathsf{OZG}(A)$ and $\overline{v}' \in \mathcal{Z}'$. Then, there is a run of A of the form $(q, \overline{v}) \xrightarrow{\sigma} (q', \overline{v}')$ such that $\overline{v} \in \mathcal{Z}$.*

2.8. Making zone graphs finite

The reachability problem for timed automata is reduced to the reachability in the offset zone graph of timed automata (see Definition 2.34). However, the zone graph of a timed automaton may not be finite. Therefore, an algorithm that decides the reachability problem for a timed automaton by exploring its zone graph is not guaranteed to terminate. To ensure termination of this algorithm, some finite truncation of this zone graph is needed.

We define an *abstraction operator*, which is a function mapping sets of valuations to sets of valuations [BBLP06, DT98, HSW12, GMS19, BBFL03].

Definition 2.35 (Abstraction operator [BBFL03])**.** An abstraction operator $\mathfrak{a} : \mathcal{P}(\mathbb{R}^{\widetilde{X}'}_{\geq 0}) \to \mathcal{P}(\mathbb{R}^{\widetilde{X}'}_{\geq 0})$, where $\widetilde{X}' = \widetilde{X} \cup \{t\}$, is a function from sets of offset valuations to sets of offset valuations such that $W \subseteq \mathfrak{a}(W)$ and $\mathfrak{a}(\mathfrak{a}(W)) = \mathfrak{a}(W)$, where W is a set of offset valuations. An abstraction operator is *finite*, if its range is finite.

Remark. *Abstraction operators, as found in the standard literature on timed automata [BBFL03, DT98, HSW12], are normally defined for standard valuations. However, since our work mainly deals with offset valuations, we adapt the definitions to the offset setting.*

An abstraction operator $\mathfrak{a}$ defines an abstract semantics, referred to as the simulation graph of the automaton based on $\mathfrak{a}$, whose definition follows.

Definition 2.36 (Simulation graph based on an abstraction)**.** Given a timed automaton A, and an abstraction operator $\mathfrak{a}$, the simulation graph of A based on $\mathfrak{a}$, denoted by $\mathsf{SG}^{\mathfrak{a}}(A)$, is a transition system whose

- states are of the form (q, W), where q is a state of A and W is a set of offset valuations.
- initial state is of the form (q_0, W_0), where $W_0 = \mathfrak{a}(\mathcal{Z}_0)$, where $\mathcal{Z}_0$ is the initial zone, i.e., $\mathcal{Z}_0 = \{\overline{v}_0 + \delta \mid \delta \in \mathbb{R}_{\geq 0}\}$, where $\overline{v}_0$ is an initial valuation.

- transition relation is:

$$(q, W) \xrightarrow{t}_{\mathfrak{a}} (q', \mathfrak{a}(W'))$$

if $W = \mathfrak{a}(W)$ and $W' = \{\overline{v}' \mid \exists\ \overline{v} \in W, \exists\ \delta \in \mathbb{R}_{\geq 0}$ s.t. $(q, \overline{v}) \xrightarrow{t} \xrightarrow{\delta} (q', \overline{v}')\}$ is non-empty.

We now define when $\mathsf{SG}^{\mathfrak{a}}(A)$ is said to be *sound* and *complete* for a timed automaton A and an abstraction operator $\mathfrak{a}$.

Soundness: $\mathsf{SG}^{\mathfrak{a}}(A)$ is said to be sound if for every path $(q_0, W_0) \xrightarrow{\rho}_{\mathfrak{a}} (q, W)$ in $\mathsf{SG}^{\mathfrak{a}}(A)$ where $W_0 = \mathfrak{a}(\mathcal{Z}_0)$ is the abstraction of the initial zone, there are valuations $\overline{v}_0 \in \mathcal{Z}_0$ and $\overline{v} \in W$ such that $(q_0, \overline{v}_0) \xrightarrow{\rho} (q, \overline{v})$.

Completeness: $\mathsf{SG}^{\mathfrak{a}}(A)$ is said to be complete if for every run of A of the form $(q_0, \overline{v}_0) \xrightarrow{\sigma} (q, \overline{v})$, there is a path in $\mathsf{SG}^{\mathfrak{a}}(A)$ of the form $(q_0, W_0) \xrightarrow{\sigma}_{\mathfrak{a}} (q, W)$, such that $\overline{v} \in W$.

From the definitions of soundness and completeness of $\mathsf{SG}^{\mathfrak{a}}(A)$, it is clear that if we can compute an abstraction operator $\mathfrak{a}$ such that $\mathsf{SG}^{\mathfrak{a}}(A)$ is sound, complete and finite, then it is sufficient to explore $\mathsf{SG}^{\mathfrak{a}}(A)$ to decide the reachability problem for A. In the next section, we discuss ways to design such abstraction operators which give rise to sound, complete and finite simulation graphs.

Abstractions from simulations

A convenient way to construct abstraction operators is using simulation relations between valuations. We formalize this notion by using the idea of a time-abstract simulation (see Definition 2.18).

An abstraction $\mathfrak{a}$ based on $\preceq_A$ is defined as

$$\mathfrak{a}(W) = \{\overline{v} \mid \exists \overline{v}' \in W \text{ with } \overline{v} \preceq_A \overline{v}'\}$$

Next, in Lemmas 2.20 and 2.21, we show some properties of the transitions of the simulation graph of a timed automaton. We then use these properties in Lemma 2.23 to prove that the simulation graph of a timed automaton is sound and complete.

Lemma 2.20. *Let W be an arbitrary set of valuations. If $(q, W) \xrightarrow{b} (q_1, W_1)$ and $(q, \mathfrak{a}(W)) \xrightarrow{b}_{\mathfrak{a}} (q_1, W_1')$, then $W_1' = \mathfrak{a}(W_1)$.*

Proof. Recall that $(q, \mathfrak{a}(W)) \xrightarrow{b}_{\mathfrak{a}} (q_1, W_1')$ stands for $(q, \mathfrak{a}(W)) \xrightarrow{b} (q_1, W_1'')$ and $W_1' = \mathfrak{a}(W_1'')$.

To show $\mathfrak{a}(W_1) \subseteq W_1'$ we observe that $W \subseteq \mathfrak{a}(W)$ by the definition of the abstraction. Then, $W_1 \subseteq W_1''$ by the definition of the transition. Finally, $\mathfrak{a}(W_1) \subseteq \mathfrak{a}(W_1'') = W_1'$ by the monotonicity of the abstraction.

To show that $W_1' \subseteq \mathfrak{a}(W_1)$, consider $\overline{v}_1' \in W_1' = \mathfrak{a}(W_1'')$. By definition, there is $\overline{v}_1'' \in W_1''$ such that $\overline{v}_1' \preceq_A \overline{v}_1''$. From the definition of transition

$(q, \mathfrak{a}(W)) \xrightarrow{b} (q_1, W_1'')$, we get a valuation $\overline{v}' \in \mathfrak{a}(W)$ such that $(q, \overline{v}') \xrightarrow{\delta} \xrightarrow{b} (q', \overline{v}_1'')$ for some δ, Since $\overline{v}' \in \mathfrak{a}(W)$, there is $\overline{v} \in W$ with $\overline{v}' \preceq_A \overline{v}$ and a transition $(q, \overline{v}) \xrightarrow{\delta'} \xrightarrow{b} (q', \overline{v}_1''')$, for some δ' and some $\overline{v}_1'''$ with $\overline{v}_1'' \preceq_A \overline{v}_1'''$. We have $\overline{v}_1''' \in W_1$ by the definition of W_1. Moreover, $\overline{v}_1' \preceq_A \overline{v}_1'' \preceq_A \overline{v}_1'''$, showing that $\overline{v}_1' \in \mathfrak{a}(W_1)$. □

Using the property given in Lemma 2.20 and the definition of the simulation graph, we now show that there is a path σ in the offset zone graph to a node $(q, \mathcal{Z})$ if and only if there is a path σ to $(q, \mathfrak{a}(\mathcal{Z}))$.

Lemma 2.21. *Let $(q_0, \mathcal{Z}_0) \xrightarrow{\sigma} (q_n, \mathcal{Z}_n)$ be a path in $\mathsf{OZG}(A)$. Then, there exists a path $(q_0, W_0) \xrightarrow{\sigma}_{\mathfrak{a}} (q_n, W_n)$ in $\mathsf{SG}^{\mathfrak{a}}(A)$, such that $W_n = \mathfrak{a}(\mathcal{Z}_n)$.*

Proof. Let the path $\sigma = b \cdot \sigma'$ be as follows: $(q_0, \mathcal{Z}_0) \xrightarrow{b} (q_1, \mathcal{Z}_1) \xrightarrow{\sigma'} (q_n, \mathcal{Z}_n)$. Take $W_0 = \mathfrak{a}(\mathcal{Z}_0)$. Consider $(q_0, W_0) \xrightarrow{b}_{\mathfrak{a}} (q_1, W_1)$. By Lemma 2.20, it follows that $W_1 = \mathfrak{a}(\mathcal{Z}_1)$. Repeating the same argument with W_1 and $\mathcal{Z}_1$, we eventually get a path $(q_0, W_0) \xrightarrow{\sigma}_{\mathfrak{a}} (q_n, W_n)$ in $\mathsf{SG}^{\mathfrak{a}}(A)$ with $W_n = \mathfrak{a}(\mathcal{Z}_n)$. □

Lemma 2.22. *Let $(q_0, W_0) \xrightarrow{\sigma}_{\mathfrak{a}} (q, W)$ be a path in $\mathsf{SG}^{\mathfrak{a}}(A)$. There exists a run in A of the form $(q_0, \overline{v}_0) \xrightarrow{\sigma} (q, \overline{v})$, where $\overline{v}_0 \in W_0$ and $\overline{v} \in W$.*

Proof. Using Lemma 2.20, we know that there is a path $(q_0, \mathcal{Z}_0) \xrightarrow{\sigma} (q, \mathcal{Z})$ with $W_0 = \mathfrak{a}(\mathcal{Z}_0)$ and $W = \mathfrak{a}(\mathcal{Z})$. For every $\overline{v} \in \mathcal{Z}$, by the pre property of offset zones, there is a run of A of the form $(q_0, \overline{v}_0) \xrightarrow{\sigma} (q, \overline{v})$ for some $\overline{v}_0 \in W_0$. □

Lemma 2.23. *If $\mathfrak{a}$ is based on a time-abstract simulation for A, then $\mathsf{SG}^{\mathfrak{a}}(A)$ is sound and complete.*

Proof. The soundness of $\mathsf{SG}^{\mathfrak{a}}(A)$ follows from Lemma 2.22.

For completeness, consider a run of A of the form $(q_0, \overline{v}_0) \xrightarrow{\sigma} (q, \overline{v})$. By Lemma 2.19, we know that there is a run in $\mathsf{OZG}(A)$ of the form $(q_0, \mathcal{Z}_0) \xrightarrow{\sigma} (q, \mathcal{Z})$, such that $\overline{v} \in \mathcal{Z}$. By Lemma 2.21 we get $(q_0, W_0) \xrightarrow{\sigma} (q, W)$ in $\mathsf{SG}^{\mathfrak{a}}(A)$ such that $\mathcal{Z} \subseteq W$. Thus, we have the completeness of $\mathsf{SG}^{\mathfrak{a}}(A)$. □

Lemma 2.23 suggests that one can use the abstraction based on the coarsest simulation relation for a given automaton. However, it is known that computing the coarsest simulation relation for a given timed automaton is EXPTIME-hard [LS00]. Therefore, in the next section we explore if it is actually possible to compute some (not necessarily the coarsest) finite-time abstract simulation for a timed automaton.

Abstractions from the structure of the timed automaton

Instead of computing the coarsest relation for a given timed automaton, we look at some easily computable parameters and base the simulation relation on them. These parameters will be the maximal constants used in guards of the automaton. We then introduce a relation between offset valuations that is a time-abstract simulation for all automata whose guards are within the constants used to define the relation. This relation turns out to be easier to compute than the coarsest time-abstract simulation relation for a timed automaton [HSW12].

Behrmann et al. [BBLP06] introduced a preorder relation $\preceq_{LU}$ between standard valuations. Here, L and U are functions bounding the constants that can be used in guards. We adapt this to the offset setting to define a preorder relation $\preceq_{LU}$ between offset valuations. We then show that this preorder relation is actually a time-abstract simulation for every timed automaton with constants in guards bounded by L and U. The definition of $\preceq_{LU}$ directly gives efficiently computable conditions to decide if two valuations are related by $\preceq_{LU}$.

To help us to define $\preceq_{LU}$, we now introduce the notion of LU *bounds* for a timed automaton. Consider two functions $L, U : X \to \mathbb{N} \cup \{-\infty\}$ assigning to each clock a bound that is a natural number or $-\infty$. Intuitively, $-\infty$ means that the clock is not used. We say that a guard g *conforms* to L and U if

- for every upper bound constraint in g of the form $x < c$ or $x \leq c$, we have $c \leq U(x)$.
- for every lower bound constraint in g of the form $x > c$ or $x \geq c$, we have $c \leq L(x)$.

Observe that if L is not bigger than L' in the pointwise ordering, meaning that $L(x) \leq L'(x)$ for every clock x, then, when g conforms to L, it also conforms to L'. Similarly for U.

Definition 2.37 (LU bounds for an automaton)**.** Given a timed automaton A, the LU bounds for A are the smallest functions L and U, such that all the guards on all the transitions of A conform to L and U.

We recall the definition of LU-preorder between standard valuations, as given in [BBLP06].

Definition 2.38 (LU-preorder)**.** Given two standard valuations v and v', we say $v \preceq_{LU} v'$ if for all clocks $x \in X$:

- either $v(x) = v'(x)$,
- or $L(x) < v'(x) \leq v(x)$,

- or $U(x) < v(x) \leq v'(x)$.

We now define LU-preorder between offset valuations by adapting Definition 2.38 to the offset setting,using the translation from offset valuations to standard valuations.

Definition 2.39 (LU-preorder for offset valuations)**.** Let $\overline{v}$ and $\overline{v}'$ be offset valuations. $\overline{v} \preceq_{LU} \overline{v}'$ if and only if for all $x \in X$

- either $\overline{v}(t) - \overline{v}(\widetilde{x}) = \overline{v}'(t) - \overline{v}'(\widetilde{x})$
- or $L_x < \overline{v}'(t) - \overline{v}'(\widetilde{x}) \leq \overline{v}(t) - \overline{v}(\widetilde{x})$
- or $U_x < \overline{v}(t) - \overline{v}(\widetilde{x}) \leq \overline{v}'(t) - \overline{v}'(\widetilde{x})$

Lemma 2.24. *$\overline{v} \preceq_{LU} \overline{v}'$ iff $\mathsf{std}(\overline{v}) \preceq_{LU} \mathsf{std}(\overline{v}')$.*

Proof. The proof follows from Definition 2.11 and Definition 2.38. □

Lemma 2.25. *If A is a timed automaton whose guards conform to LU, then the relation $\preceq_{LU}$ is a time-abstract simulation for A.*

Proof. Let $\overline{v} \preceq_{LU} \overline{v}'$. Consider a delay of δ from the valuation $\overline{v}$. It is easy to see that $\overline{v} + \delta \preceq_{LU} \overline{v}' + \delta$, since the differences of the form $t - \widetilde{x}$ increase by the same value in both $\overline{v} + \delta$ and $\overline{v}' + \delta$.

Next, consider an action transition a from the state $(q, \overline{v})$ with guard g and reset R. We show that if $\overline{v} \models g$, then $\overline{v}' \models g$. Let g be of the form $x \leq c$. For an offset clock, there are three possibilities for the difference $t - \widetilde{x}$.

- $\overline{v}(t - \widetilde{x}) = \overline{v}'(t - \widetilde{x})$. It is clear that $\overline{v} \models g$ iff $\overline{v}' \models g$.
- $L_x < \overline{v}'(t - \widetilde{x}) \leq \overline{v}(t - \widetilde{x})$. Let $\overline{v} \models g$. This means that $\overline{v}(t - \widetilde{x}) \leq c$. But since $\overline{v}'(t - \widetilde{x}) \leq \overline{v}(t - \widetilde{x})$, it follows that $\overline{v}'(t - \widetilde{x}) \leq c$. Hence, $\overline{v}' \models g$.
- $U_x < \overline{v}(t - \widetilde{x}) \leq \overline{v}'(t - \widetilde{x})$. Since $c \leq U_x$, we have $c < \overline{v}(t - \widetilde{x})$. Therefore, $\overline{v} \models g$ is never true.

Suppose that g is of the form $x \geq c$. Again, there are three possibilities for the difference $t - \widetilde{x}$.

- $\overline{v}(t - \widetilde{x}) = \overline{v}'(t - \widetilde{x})$. It is clear that $\overline{v} \models g$ iff $\overline{v}' \models g$.
- $L_x < \overline{v}'(t - \widetilde{x}) \leq \overline{v}(t - \widetilde{x})$. It may be observed that $\overline{v}' \models g$ is always true, since $\overline{v}'(t - \widetilde{x}) > L_x \geq c$.
- $U_x < \overline{v}(t - \widetilde{x}) \leq \overline{v}'(t - \widetilde{x})$. $\overline{v} \models g$ implies that $\overline{v}(t - \widetilde{x}) \geq c$. Since $\overline{v}'(t - \widetilde{x}) \geq \overline{v}(t - \widetilde{x})$, we can observe that $\overline{v}'(t - \widetilde{x}) \geq c$ and therefore $\overline{v}' \models g$.

Since $\overline{v}' \models g$ is true whenever $\overline{v} \models g$ is true, it is clear that $(q, \overline{v}')$ can execute the action a if $(q, \overline{v})$ can execute a. Let $(q, \overline{v}) \xrightarrow{a} (q', \overline{v}_1)$ and $(q, \overline{v}') \xrightarrow{a} (q', \overline{v}'_1)$. If a clock x is reset, then the differences $t - \widetilde{x}$ and $\widetilde{x} - t$ get set to 0 in the resultant valuation. This implies that $t - \widetilde{x}$ and $\widetilde{x} - t$ get set to 0 in $\overline{v}_1$ and $\overline{v}'_1$. Since resets leave the other differences unaltered, it is clear that $\overline{v}_1 \preceq_{LU} \overline{v}'_1$. □

In the following we will use $\mathfrak{a}_{\preccurlyeq_{LU}}$, the abstraction operator based on the time-abstract simulation relation $\preceq_{LU}$. For the sake of completeness, we formally define it below.

Definition 2.40 ($\mathfrak{a}_{\preccurlyeq_{LU}}$ abstraction)**.** Given bounds L and U for a network of timed automata, for a set of valuations W we define:

$$\mathfrak{a}_{\preccurlyeq_{LU}}(W) = \{\overline{v} \mid \exists \overline{v}' \in W \text{ s.t. } \overline{v} \preceq_{LU} \overline{v}'\}$$

Next, we define when a zone is $\mathfrak{a}_{\preccurlyeq_{LU}}$-simulated by another zone.

Definition 2.41 ($\mathfrak{a}_{\preccurlyeq_{LU}}$ simulation for zones)**.** Given zones $\mathcal{Z}$ and $\mathcal{Z}'$, we say $\mathcal{Z}$ is $\mathfrak{a}_{\preccurlyeq_{LU}}$ simulated by $\mathcal{Z}'$ if $\mathcal{Z} \subseteq \mathfrak{a}_{\preccurlyeq_{LU}}(\mathcal{Z}')$.

We now proceed to show that this abstraction operator has finite range when applied over zones. This is done using the notions of region and neighbourhood introduced earlier in Definitions 2.15 and 2.16.

Lemma 2.26. *Let M be the maximal value of L and U functions. If $v \preceq_{LU} v'$ and $v_1 \in [v]^M$, then there is a valuation $v'_1 \in \mathsf{nbd}(v')$ such that $v_1 \preceq_{LU} v'_1$. In particular $v'_1 \in [v']^M$.*

Proof. For simplicity we prove the lemma for standard valuations. The proof for offset valuations follows the same lines, but needs to talk about differences between $\overline{v}(t)$ and $\overline{v}(x)$ which complicates the writing.

We partition the set of clocks into three subsets:

- big clocks: are clocks x such that $v(x) > M$;
- fixed clocks: are clocks z such that $v(z) < \min(L(x), U(x))$;
- flexible clocks: are clocks y that are none of the above.

Consider a valuation $v_1 \in [v]^M$. We need to find a valuation $v'_1 \in \mathsf{nbd}(v')$. For big clocks, we set $v'_1(x) = v'(x)$. Similarly, for fixed clocks, $v'_1(z) = v'(z)$. For the flexible clocks, we need to work bit more. The integer part of flexible clocks is determined: requirement $v'_1 \in \mathsf{nbd}(v')$ implies that the integer part of $v'_1(y)$ must be the same as the integer part of $v'(y)$ for every clock.

We need to define the values of the fractional parts of flexible clocks. For this, we list the flexible clocks in an arbitrary order, say $y_1, \ldots, y_l$ and find the fractional part of the v' value of one flexible clock after the

other. Observe that the order of fractional parts of fixed and flexible clocks is the same in v and v_1, because of $v_1 \in [v]^M$. Moreover the order of fractional parts of fixed clocks in v' is by definition the same as in v. So, the order of fractional parts of $v(z_1), \ldots, v(z_k), v(y_1), \ldots, v(y_l)$ is the same as of $v_1(z_1), \ldots, v_1(z_k), v_1(y_1), \ldots, v_1(y_l)$.

We proceed by induction on $i = 0, \ldots, l$. Suppose that the order of fractional parts of $v(z_1), \ldots, v(z_k), v(y_1), \ldots, v(y_i), v'(y_1), \ldots, v'(y_i)$ is the same as that of $v_1(z_1), \ldots, v_1(z_k), v_1(y_1), \ldots, v_1(y_i), v'_1(y_1), \ldots, v'_1(y_i)$. The base case of $i = 0$ is true by the above paragraph. We find the value of the fractional part of $v'_1(y_{i+1})$ that would make this statement true also for y_{i+1}. This is quite direct. Suppose that $v'(y_{i+1})$ is between $v(z_a)$ and $v'(y_b)$ for $b \leq i$. Then, we choose for $v'(y_{i+1})$ the fractional value that will put it in between $v(z_a)$ and $v'(y_b)$. This is always possible as real numbers are dense. There is a special case when $v'(y_{i+1})$ is equal to some of the other values, say $v(z_a)$, but in that case, we just put $v'_1(y_{i+1}) = v(z_a)$.

Thus, we have constructed a valuation $v'_1 \in \mathsf{nbd}(v')$ such that the order of the fractional parts in $v(z_1), \ldots, v(z_k), v(y_1), \ldots, v(y_l), v'(y_1), \ldots, v'(y_l)$ is the same as in $v(z_1), \ldots, v(z_k), v_1(y_1), \ldots, v_1(y_l), v'_1(y_1), \ldots, v'_1(y_l)$.

We claim that $v' \preceq_{LU} v'_1$. For this, we look at the definition of the $\preceq_{LU}$ preorder (Definition 2.38). We observe that from $v' \in [v]^M$, we can infer that if $v(x) > L(x)$, then $v_1(x) > L(x)$; and analogously for $U(x)$.

If x is a big clock, then $v(x) > L(x), U(x)$. This means that $v'(x) > L(x)$. By the observation from the previous paragraph $v_1(x) > L(x), U(x)$. Since $v'_1 \in \mathsf{nbd}(v')$, we have $v'_1 > L(x)$, and we are done.

If z is a fixed clock, then $v(z) = v'(z)$, and $v_1(z) = v'_1(z)$; so we are done also in this case.

If y is a flexible clock, then we have two cases. If $v(y) > L(y)$, then $L(y) \leq v'(y) \leq v(y)$. From $v' \in [v]^M$, we deduce that the integer parts of $v'(y)$ and $v(y)$ are the same. From $v'_1 \in \mathsf{nbd}(v')$, we deduce that the integer parts of $v'_1(y)$ and $v'(y)$ are the same. Thus, we have $L(y) \leq v_1(y), v'_1(y)$. If the integer parts of $v'(y)$ and $v(y)$ are different, then we get as desired $v_1(y) \leq v'_1(y)$. If they are the same, then we get $v_1(y) \leq v'_1(y)$ because we have taken care of the order of fractional parts when constructing v'_1. The argument when $v(y) > U(x)$ is analogous. □

Lemma 2.9 which gives a bijection between standard regions and offset regions and Lemma 2.24 which gives the relation between $\preceq_{LU}$ equivalence for standard valuations and offset valuations allow us to state Lemma 2.26 for offset valuations also.

Corollary 2.2. *Let M be the maximal value of L and U functions. If $\overline{v} \preceq_{LU} \overline{v}'$ and $\overline{v}_1 \in [\overline{v}]^M$, then there is a valuation $\overline{v}'_1 \in \mathsf{nbd}(\overline{v}')$ such that $\overline{v}_1 \preceq_{LU} \overline{v}'_1$. In particular $\overline{v}'_1 \in [\overline{v}']^M$.*

Lemma 2.27. *If W is closed under region equivalence relation $\equiv_M$, for M the maximum of L and U functions, then so is $\mathfrak{a}_{\preccurlyeq LU}(W)$.*

Proof. We show that for arbitrary W as in assumption of the lemma, if a valuation $\overline{v} \in \mathfrak{a}_{\preccurlyeq LU}(W)$, and $\overline{v}_1 \in [\overline{v}]^M$, then $\overline{v}_1 \in \mathfrak{a}_{\preccurlyeq LU}(W)$ (see Definition 2.15).

As $\overline{v} \in \mathfrak{a}_{\preccurlyeq LU}(W)$, there is $\overline{v}' \in W$ such that $\overline{v} \preceq_{LU} \overline{v}'$. By Corollary 2.2, we know that there exists a valuation $\overline{v}'_1 \in \mathsf{nbd}(\overline{v}')$ such that $\overline{v}_1 \preceq_{LU} \overline{v}'_1$. Since W is neighborhood closed, $\overline{v}'_1 \in W$, and in consequence $\overline{v}_1 \in \mathfrak{a}_{\preccurlyeq LU}(W)$. □

From the discussion above it is clear that $\mathfrak{a}_{\preccurlyeq LU}(W)$ is a union of M-regions, for every zone W. From Lemma 2.10, we know that the number of M-regions is finite. This implies that the number of distinct sets $\mathfrak{a}_{\preccurlyeq LU}(W)$ is finite. Thus, $\mathfrak{a}_{\preccurlyeq LU}(W)$ has finite range.

Lemma 2.28. $\mathsf{SG}^{\mathfrak{a}_{\preccurlyeq \mathsf{LU}}}(A)$ *is finite, sound and complete.*

Proof. The soundness and completeness of $\mathsf{SG}^{\mathfrak{a}_{\preccurlyeq \mathsf{LU}}}(A)$ follows from Lemma 2.23. The finiteness of $\mathsf{SG}^{\mathfrak{a}_{\preccurlyeq \mathsf{LU}}}(A)$ follows from Lemma 2.27, which says that $\mathfrak{a}_{\preccurlyeq LU}$. □

Since $\mathsf{SG}^{\mathfrak{a}_{\preccurlyeq \mathsf{LU}}}(A)$ is an abstraction of the zone graph of A which is sound, complete and finite, checking the reachability of a state of the timed automaton A can be done by exploration of $\mathsf{SG}^{\mathfrak{a}}(A)$.

2.9. Standard reachability algorithm

In this section, we present the standard reachability algorithm for timed automata using the ideas introduced in this chapter. The algorithm takes as input a timed automaton $A := (Q, \Sigma, X, T, q_0, F)$, computes its LU bounds (Definition 2.37), and constructs a simulation graph $\mathsf{SG}^{\mathfrak{a}}(A)$ as in Definition 2.36 using the abstraction $\mathfrak{a}_{\preccurlyeq LU}$ from Definition 2.40.

Thanks to Lemma 2.20, the algorithm can store pairs $(q, \mathcal{Z})$ and not $(q, \mathfrak{a}_{\preccurlyeq LU}(\mathcal{Z}))$. This is important as $\mathfrak{a}_{\preccurlyeq LU}(\mathcal{Z})$ may not be a zone, so we do not have an efficient way to store $\mathfrak{a}_{\preccurlyeq LU}(\mathcal{Z})$. What we need instead is an efficient method to test $\mathfrak{a}_{\preccurlyeq LU}(\mathcal{Z}) \subseteq \mathfrak{a}_{\preccurlyeq LU}(\mathcal{Z}')$. Observe that by monotonicity, this is equivalent to a test $\mathcal{Z} \subseteq \mathfrak{a}_{\preccurlyeq LU}(\mathcal{Z}')$. We prefer a more compact notation $\mathcal{Z} \sqsubseteq^{\mathfrak{a}}_{LU} \mathcal{Z}'$ instead of $\mathcal{Z} \subseteq \mathfrak{a}_{\preccurlyeq LU}(\mathcal{Z}')$.

Algorithm 1 starts the exploration from the initial node $(q_0, \mathcal{Z}_0)$. It keeps a list Visited of all the nodes constructed by the algorithm, and a list Waiting of nodes that have been constructed but whose successors have not been computed yet. In the beginning, both lists contain the initial node $(q_0, \mathcal{Z}_0)$. In each iteration, the algorithm removes a node from Waiting and adds to Waiting all its successors. There are two exceptions to this behavior. First,

if an accepting state is reached, then the algorithm terminates. Second, if a node to be inserted to Waiting is $\sqsubseteq^{\mathfrak{a}}_{LU}$ smaller than some already visited node, then there is no point in adding it to Waiting.

We also optimize the algorithm further, by adding the following step: whenever a node $(q, \mathcal{Z})$ is added to Waiting and Visited, we remove any node $(q, \mathcal{Z}'')$ in Waiting and Visited such that $\mathcal{Z}'' \sqsubseteq^{\mathfrak{a}}_{LU} \mathcal{Z}$. We will show that this does not affect the correctness of the algorithm.

Algorithm 1 Reachability algorithm for a timed automaton

Input : Timed automaton $A := (Q, \Sigma, X, T, q_0, F)$.
Output : true iff A has a run reaching an accepting state.

1: Set Waiting = Visited := $\{(q_0, \mathcal{Z}_0)\}$
2: **if** q_0 is accepting **then return** true
3: **while** Waiting $\neq \emptyset$ **do**
4: remove some $(q, \mathcal{Z})$ from Waiting
5: **for** all $(q', \mathcal{Z}')$ s.t. $(q, \mathcal{Z}) \xrightarrow{a} (q', \mathcal{Z}')$ for some a **do**
6: **if** q' is accepting **then**
7: **return** true
8: **else if** $\exists\ (q', \mathcal{Z}'') \in$ Visited s.t. $\mathcal{Z}' \sqsubseteq^{\mathfrak{a}}_{LU} \mathcal{Z}''$ **then**
9: Skip
10: **else**
11: **for** $(q', \mathcal{Z}'') \in$ Visited **do**
12: **if** $\mathcal{Z}'' \sqsubseteq^{\mathfrak{a}}_{LU} \mathcal{Z}'$ **then**
13: Remove $(q', \mathcal{Z}'')$ from Visited and Waiting
14: Add $(q', \mathcal{Z}')$ to Waiting and Visited
15: **return** false

Correctness of the algorithm

We will now prove that Algorithm 1 is correct. The soundness of the algorithm is given by Lemma 2.30, while the completeness follows from Lemma 2.33.

Lemma 2.29. *If a node $(q, \mathcal{Z})$ is in the list* Waiting *of Algorithm 1, then A has a run $(q_0, \overline{v}_0)$ to $(q, \overline{v})$, for some valuation $\overline{v} \in \mathcal{Z}$.*

Proof. Directly from the algorithm, we see that if $(q, \mathcal{Z})$ is in Waiting, then there is a sequence of actions σ such that $(q_0, \mathcal{Z}_0) \xrightarrow{\sigma} (q, \mathcal{Z})$. The claim of the lemma follows from Lemma 2.22. □

Lemma 2.30. ***(Soundness)*** *If Algorithm 1 returns* true*, then A has an accepting run.*

Proof. Algorithm 1 can return true by either executing line 2 or line 7. Line 2 is only executed in the special case when the initial state q_0 of A

is an accepting state - we know that in this case, the claim is vacuously true. Next, consider the case when line 7 is executed. We can infer from the algorithm that this happens only when a node $(q, \mathcal{Z})$ has just been removed from Waiting, and $(q, \mathcal{Z}) \xrightarrow{a} (q', \mathcal{Z}')$ is such that q' is an accepting state. By Lemmas 2.29 and 2.22, we know that A has an accepting run. □

Lemma 2.31. *If a node $(q, \mathcal{Z})$ is added to* Visited *at some step of Algorithm 1, then at each step afterwards there is a node $(q, \mathcal{Z}')$ in* Visited *such that $\mathcal{Z} \sqsubseteq^{\mathfrak{a}}_{LU} \mathcal{Z}'$.*

Proof. Suppose that a node $(q, \mathcal{Z})$ was added to Visited at some stage of Algorithm 1. If $(q, \mathcal{Z})$ is in Visited until the termination of the algorithm, then we are done. Suppose not. We can see that line 13 is the only step of the algorithm that removes a node $(q, \mathcal{Z})$ from Visited. But in this case, we can observe that another node $(q, \mathcal{Z}')$ is added to Visited such that $\mathcal{Z} \sqsubseteq^{\mathfrak{a}}_{LU} \mathcal{Z}'$. Again, if $(q, \mathcal{Z}')$ is removed at a later point in the algorithm, then by the same argument, we know that another state $(q, \mathcal{Z}'')$ such that $\mathcal{Z}' \sqsubseteq^{\mathfrak{a}}_{LU} \mathcal{Z}''$ is added to Visited. □

Lemma 2.32. *Let σ be path in* OZG(A)*:*

$$(q_0, \mathcal{Z}_0) \xrightarrow{a_1} (q_1, \mathcal{Z}_1) \xrightarrow{a_2} \cdots (q_{n-1}, \mathcal{Z}_{n-1}) \xrightarrow{a_n} (q_n, \mathcal{Z}_n).$$

If Algorithm 1 does not return true *at the termination, then for every $0 \leq i \leq n$ there exists a state $(q_i, \mathcal{Z}'_i)$ in the set* Visited *such that $\mathcal{Z}_i \sqsubseteq^{\mathfrak{a}}_{LU} \mathcal{Z}'_i$.*

Proof. Suppose that Algorithm 1 does not return true. We will give a proof by induction on the number of transitions in σ.

Base case: The node $(q_0, \mathcal{Z}_0)$ is added to Visited in Line 1 of the algorithm. By Lemma 2.31, until the end of the execution of the algorithm, there is some node $(q_0, \mathcal{Z}') \in$ Visited such that $\mathcal{Z}_0 \sqsubseteq^{\mathfrak{a}}_{LU} \mathcal{Z}'$.

Induction step: By induction hypothesis, there exists a node $(q_i, \mathcal{Z}'_i)$ in Visited, such that $\mathcal{Z}_i \sqsubseteq^{\mathfrak{a}}_{LU} \mathcal{Z}'_i$. Assume w.l.o.g. that $(q_i, \mathcal{Z}'_i)$ is a maximal node in Visited with respect to $\sqsubseteq^{\mathfrak{a}}_{LU}$ ordering. Since $(q_i, \mathcal{Z}'_i)$ is in Visited, we know that it must have also been in Waiting at some stage of the algorithm. Further, since we know that the algorithm terminated when Waiting is empty, at some stage of the algorithm, this node $(q_i, \mathcal{Z}'_i)$ must have been removed from Waiting.

From the run σ, we know that $(q_i, \mathcal{Z}_i) \xrightarrow{a_{i+1}} (q_{i+1}, \mathcal{Z}_{i+1})$. From Lemma 2.25, we know that $\sqsubseteq^{\mathfrak{a}}_{LU}$ is a simulation relation. So, $(q_i, \mathcal{Z}'_i) \xrightarrow{a_{i+1}} (q_{i+1}, \mathcal{Z}'_{i+1})$, for some $\mathcal{Z}'_{i+1}$ such that $\mathcal{Z}_{i+1} \sqsubseteq^{\mathfrak{a}}_{LU} \mathcal{Z}'_{i+1}$. If this node is added to Visited, then by Lemma 2.31, we know that until the end of the execution of the algorithm, there is some node $(q_{i+1}, \mathcal{Z}''_{i+1}) \in$ Visited such that $\mathcal{Z}_{i+1} \sqsubseteq^{\mathfrak{a}}_{LU} \mathcal{Z}''_{i+1}$. In this case, we are done.

Now, suppose that $(q_{i+1}, \mathcal{Z}'_{i+1})$ is not added to Visited. This could happen due to two reasons:

- $(q_{i+1}, \mathcal{Z}'_{i+1})$ is an accepting node. In this case, Algorithm 1 would have terminated by returning true. But by our assumption, this is not the case.
- There exists $(q_{i+1}, \mathcal{Z}''_{i+1})$ in Visited such that $\mathcal{Z}'_{i+1} \sqsubseteq^{\mathfrak{a}}_{LU} \mathcal{Z}''_{i+1}$. Since $\mathcal{Z}_{i+1} \sqsubseteq^{\mathfrak{a}}_{LU} \mathcal{Z}'_{i+1}$, by transitivity of $\sqsubseteq^{\mathfrak{a}}_{LU}$, we can also conclude in this case. □

Lemma 2.33. ***(Completeness)*** *If* $\mathsf{OZG}(A)$ *has a reachable accepting node, then Algorithm 1 returns* true.

Proof. Suppose that Algorithm 1 terminated by returning false. Then, by Lemma 2.32, if there is a run to an accepting node $(q, \mathcal{Z})$ of $\mathsf{OZG}(A)$, it follows that $(q, \mathcal{Z})$ is in Visited when the algorithm terminates. But this is not possible as the accepting state is never added to Visited. □

From Lemma 2.30 and Lemma 2.33, we can infer that Algorithm 1 is correct. The termination of the algorithm is guaranteed by the finiteness of $\mathfrak{a}_{\preccurlyeq_{LU}}$, given by Lemma 2.27.

2.10. Partial order reduction

Concurrent systems, such as a network of processes, have several components operating in parallel. We first set up some basic definitions and notation before motivating the key ideas of partial order reduction techniques.

Definition 2.42 (Process)**.** A *process* is a tuple $\langle S, \Sigma, s_0, \to, F\rangle$, where S is a set of states, Σ is a finite alphabet of actions, $\to \subseteq S \times \Sigma \times S$ is a transition relation between states, $s_0 \in S$ is an initial state and $F \subseteq S$ is a set of accepting states. We write $s \xrightarrow{a} s'$ to denote that $(s, a, s') \in \to$. We sometimes refer to $\to$ as the *successor relation* of the process.

Given a process A_p and a state s of A_p, we say that action a is *enabled* in state s if there exists a transition $s \xrightarrow{a} s'$.

Definition 2.43 (Run of the process)**.** A *run* of the process A_p from a state s is a sequence of transitions starting in s: $s \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \cdots \xrightarrow{a_n} s_n$. We denote it by $s \xrightarrow{\sigma} s_n$ where $\sigma = a_1 \cdot a_2 \cdots a_n$ is a sequence of actions. We say that a run is accepting if the final state of the run is an accepting state.

Definition 2.44 (Network of processes)**.** Let $\Sigma_1, \Sigma_2, \ldots, \Sigma_k$ be a collection of finite alphabets, not necessarily disjoint. A *network of processes* is a tuple $\langle A_1, A_2, \ldots, A_k\rangle$ where each A_i is a process given by $(S_i, \Sigma_i, s_i^0, \to_i, F_i)$.

For each action $a \in \Sigma$, we define the domain of the action a as the set of processes participating in that action. Formally, $\mathsf{dom}(a) = \{p \mid a \in \Sigma_p\}$.

We refer to the actions whose domain contains more than one process as *synchronization actions*. All other actions are referred to as *local actions*.

Definition 2.45 (Semantics of a network)**.** The semantics of a network of processes $\mathcal{N}$ is given by a transition system $T_{\mathcal{N}} = (S_\times, \bigcup_{i=1}^{i=k} \Sigma_i, s^0_\times, \Rightarrow, F_\times)$ where

- the set of states $S_\times = S_1 \times S_2 \times \cdots \times S_k$,
- the initial state $s^0_\times = (s^0_1, s^0_2, \ldots, s^0_k)$,
- the set of final states $F_\times = \{\ (s_1, s_2, \ldots, s_k) \mid s_i \in F_i \text{ for all } i\ \}$,
- there is a transition $(s_1, s_2, \ldots, s_k) \stackrel{a}{\Longrightarrow} (s'_1, s'_2, \ldots, s'_k)$ if
  - for all $i \in \mathsf{dom}(a)$, action a is enabled in s_i
  - $s_i \stackrel{a}{\rightarrow}_i s'_i$ for all $i \in \mathsf{dom}(a)$, and $s'_i = s_i$ for all $i \notin \mathsf{dom}(a)$.

For each state s of $T_{\mathcal{N}}$, let $\mathsf{Act}(s)$ denote the set of transitions enabled in s. Likewise, let $\mathsf{Act}_i(s)$ denote the set of transitions enabled from s in process A_i.

Verification of a network $\mathcal{N}$ is usually carried out by the exploration of the transition system $T_{\mathcal{N}}$ modelling the semantics of the network. A state of the transition system $T_{\mathcal{N}}$ is a tuple containing a state of each component of $\mathcal{N}$. This implies that the size of the state space of $T_{\mathcal{N}}$ is the product of the sizes of the state spaces of the components of $\mathcal{N}$. As a consequence, the size of the transition system modelling the semantics of the network grows exponentially with respect to the number of components of the network. This dramatic increase in state space is commonly referred to as *state-space explosion*.

State-space explosion means that verification of a network by exhaustive exploration of a transition system modelling the network is not a scalable solution [CKNZ11]. Various solutions have been proposed to overcome the challenge posed by state-space explosion [CG18, CGJ+03]. Partial order reduction is one such technique that tries to reduce the size of the state space of the transition system that needs to be explored in order to carry out the verification of a system.

We now examine the underlying reasons for state-space explosion in networks of processes. Consider the network of processes $\mathcal{N}_1$ depicted in Figure 2.8. In this case, the transition system that is used to study $\mathcal{N}_1$ is the product automaton of the network, denoted as $A_{\mathcal{N}_1}$. Observe that the product automata of a network contains all possible orderings of concurrent actions in the network. For example, in $A_{\mathcal{N}_1}$ shown in Figure 2.8, there are 3! paths that go from the initial state (p_0, q_0, r_0) to the state (p_1, q_1, r_1).

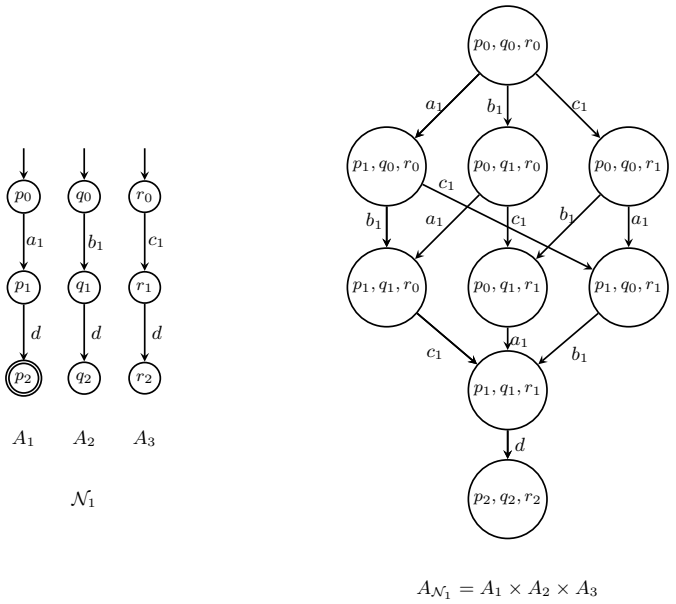


Figure 2.8: A network of processes

These paths are essentially different interleavings of the actions a_1, b_1 and c_1 - in other words, they are actions a_1, b_1 and c_1, executed in different orders. In general, if we consider a sequence of n concurrent actions each belonging to a different process, then there are $n!$ different interleavings of this sequence, one per each possible ordering of these actions. Thus, to summarize, the fundamental reason for state-space explosion is that all the orderings of concurrent actions in the network are feasible in the transition system modelling the network.

However, the exploration of all the interleavings of concurrent actions may not be necessary for the verification of a property. For instance, consider the transition system T_1 given in Figure 2.9. In T_1, if we are only interested in the reachability of the accepting state (p_2, q_2, r_2), and do not care about the path via which we reach this accepting state, we do not need to completely explore T_1. In this regard, all paths from (p_0, q_0, r_0) to (p_2, q_2, r_2) are equivalent. Therefore, it suffices to explore only one of these paths; for instance, the path given by red edges. We can consider a smaller transition system T_1^r that is restricted to the states in T_1 reachable via the red edges from its initial state.

This is the idea behind partial order reduction: partial order reduction techniques work by identifying a small part of the state space (referred to as the reduced transition system), whose exploration is sufficient to carry out the verification of the relevant property. The development of various partial order methods [Val89, God90, Pel93] have proven quite useful in improving the performance of verification procedures for concurrent systems, both in

terms of their running time and memory consumption.

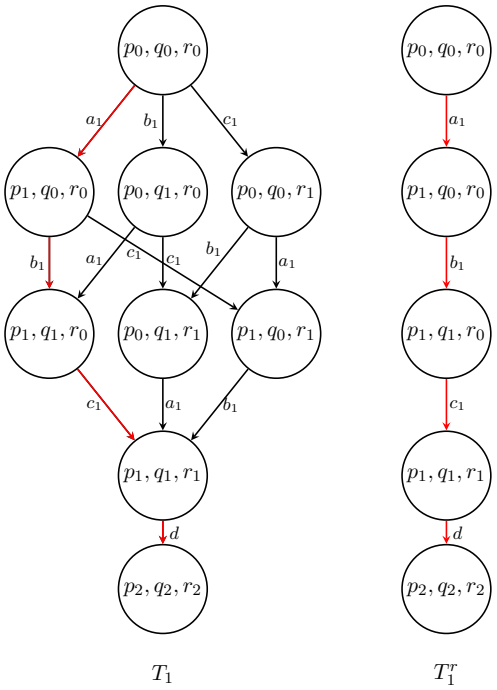


Figure 2.9: A transition system and a reduced transition system

Consider a network of n processes with k states each. If the network is such that each action of a process is completely independent of the actions of any other process of the network (in the sense that no action in the network requires participation of more than one process), then the resultant product automaton has k^n states. While applying partial order reduction, the idea is that the state space of the original transition system grows exponentially w.r.t. the number of components in the network, while that of the reduced transition system is expected to grow at a lower rate. Indeed, in the example of network given in Figure 2.8, if we increase the number of processes, the size of the reduced transition system only grows linearly w.r.t. the number of components in the network. In general, the magnitude of the reduction depends on the extent of independence between actions in the network. Nevertheless, even if we do not always observe a massive reduction in state space, the reduction obtained by partial order methods is crucial to design efficient verification procedures for networks containing several components.

In the next section, we introduce some basic principles of partial order reduction methods.

Principles of partial order reduction

The goal of this section is to explain the fundamental principles of partial order reduction methods.

We are interested in the *reachability problem* for a network of processes, defined as follows.

INPUT: a network of processes $\mathcal{N}$
OUTPUT: Yes, if there is a path from an initial state of $T_{\mathcal{N}}$ to a final state; No, otherwise

A standard solution to this problem is by an on-the-fly exploration of $T_{\mathcal{N}}$ starting from the initial state. As mentioned in the introduction of this chapter, this solution is not scalable and partial order reduction is one of the standard ways to alleviate the issue of scalability. The goal of partial order reduction is to identify a small part of $T_{\mathcal{N}}$, whose exploration is sufficient to verify the system. The idea is that executions of $T_{\mathcal{N}}$ can be classified into equivalence classes based on the property to be verified, and it is sufficient to consider only a representative from each equivalence class. For example, in the network from Figure 2.9, all the executions of T_1 that go from (p_0, q_0, r_0) to (p_2, q_2, r_2) are equivalent w.r.t. reachability. So, to answer the reachability problem for this network, it is sufficient to explore only one of these executions. This means that it is sufficient to restrict the exploration of T to the part reachable via red edges. We refer to this smaller transition system T_r whose exploration is sufficient as the *reduced transition system.*

We formalize these notions in the following definitions.

Definition 2.46 (Reduced Transition System)**.** Let $T = \langle S, \Sigma, s_0, \rightarrow, F\rangle$ be a transition system and let $\rightarrow_r \subseteq \rightarrow$ be a subset of the successor relation of T. We then define the reduced transition system induced by $\rightarrow_r$, denoted by T_r, as the restriction of T to the states reachable from s_0 using only transitions in $\rightarrow_r$. Formally, T_r is given by the tuple $\langle S_r, \Sigma, s_0, \rightarrow_r, F_r\rangle$, where S_r is the smallest set such that $s_0 \in S_r$, and if $s \in S_r$ and $s \xrightarrow{a}_r s'$ for some $a \in \Sigma$, then $s' \in S_r$. The set of accepting states of T_r is given by $F_r = S_r \cap F$. We refer to the transition relation $\rightarrow_r$ that generates T_r as the *reduced set of actions.*

Next, we specify the conditions that should be satisfied by the reduced set of actions $\rightarrow_r$ for the reduced transition system to be useful. Since we are interested in computing a reduced set that preserves the status of the reachability of the original transition system, we refer to our reduced sets as *reachability-complete sets.*

Definition 2.47 (*Reachability-complete* set)**.** Consider a transition system $T = \langle S, \Sigma, s_0, \rightarrow, F\rangle$, a transition relation $\rightarrow_r \subseteq \rightarrow$ and the reduced transition system T_r induced by $\rightarrow_r$. We say that $\rightarrow_r$ is *reachability-complete* for T,

if T_r contains an accepting state whenever an accepting state is reachable from s_0 in T.

Observe that if no accepting state is reachable in T, then the above definition puts no restrictions on T_r. Otherwise we require that at least one of the reachable accepting states is present in T_r.

Corollary 2.3. *Suppose that $T = \langle S, \Sigma, s_0, \rightarrow, F\rangle$ is a transition system, and $\rightarrow_r \subseteq \rightarrow$ that is reachability-complete for T. For T_r the reduced transition system induced by $\rightarrow_r$: T_r has an accepting run if and only if T has an accepting run.*

Next, we address the question of how such a reachability-complete set can be computed. Before doing this, it is necessary to formalize a few notions whose understanding is crucial for the design of a reachability-complete set.

Independence of actions: We say that two actions are independent in a state of a transition system, if both the orders of execution of these actions are feasible from that state, and irrespective of the order in which they are executed, the same state is reached. Recall that if an action a is feasible from a state s, then we say that a is enabled from s. We formalize this notion as follows.

Definition 2.48 (Independence of actions [KP92])**.** Let T be a transition system. Two actions a and b are said to be *independent* in T if they satisfy the following conditions for every state s of T:

Forward diamond property If a and b are enabled from s, and $s \xrightarrow{a} s_1$ and $s \xrightarrow{b} s_2$ are the respective transitions from s, then b is enabled from s_1 and a is enabled from s_2.

Diamond property If either ab or ba is feasible from s, then both ab and ba are feasible from s and moreover, both ab and ba result in the same state. The diamond property is pictorially represented by the diamond structure given in Figure 2.10.

From the definitions, it should be clear that if two actions are independent, then they commute; in other words, the order in which they are executed does not matter. This leads us to our next definition.

Definition 2.49. [Equivalent sequences [Maz86]] We say that two sequences u and w are equivalent modulo independence, denoted $u \sim w$ if w can be obtained form u by repeatedly permuting pairs of adjacent independent actions.

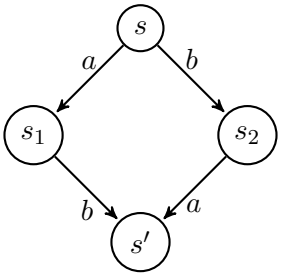


Figure 2.10: Diamond due to interleaving of actions a and b

From the definition of independence, it is clear that if two sequences are equivalent modulo independence, both of them result in the same state. Thanks to this equivalence, if we are only interested in reachability, then we do not need to consider all the sequences; rather, it is enough to consider a representative from each equivalence class.

Let T be a transition system, and let $\rightarrow_r$ be the subset of its transitions. Consider a reduced transition system T_r restricted to states that are reachable from the initial state via $\rightarrow_r$ edges. If we have the guarantee that for each execution sequence of T, there is an equivalent execution sequence in T_r, then we know that T_r has an accepting run if and only if T has an accepting run. This implies that $\rightarrow_r$ is reachability-complete and therefore, it suffices to explore the reduced transition system T_r to answer the reachability problem for T. The various partial order reduction methods, such as *Stubborn sets* by Valmari [Val90, Val89], *Ample sets* by Peled [Pel93, Pel96] and *Persistent sets* by Godefroid [God90, God96], give ways to compute such $\rightarrow_r$.

Chapter 3

Local time semantics

As discussed in Chapter 2, the most widely used approach to solve the reachability problem for networks of timed automata is by the computation and exploration of a finite truncation of the offset zone graph of the network. We point out that this approach suffers severely from state-space explosion - as the number of components in the network increases, the number of nodes in the offset zone graph of the network grows exponentially. As a result, for large networks of timed automata which contain several components, this standard approach to check reachability is not effective. In this chapter, we first analyze this explosion of state space for offset zone graphs of networks of timed automata. We show that the state-space explosion, which was already severe for untimed networks, is compounded by the timing information of clocks that is stored in the states of timed automata. As a consequence, the magnitude of explosion for timed networks is worse than that for untimed networks.

To tackle the challenge posed by state-space explosion, application of partial order reduction techniques to the offset zone graph of networks of timed automata seems to be an appealing proposition. An effective method to compute the independence relation (see Definition 2.48) between actions of the network is a crucial precondition for developing partial order reduction techniques for the verification of the network. For networks of untimed automata, disjointedness of domain gives such an effective check for independence that is easily computable. Unfortunately, we show that this idea does not work for networks of timed automata. We mention that we do not know of any effective way to compute the independence relation between actions of a network of timed automata.

As a workaround, we consider an alternate semantics for networks of timed automata, namely the *local time semantics*. The goal of this chapter is to introduce local time semantics for networks of timed automata. We will present the basic operations of local time semantics and show some basic properties of these operations. We will also discuss some properties

of runs in this semantics, using which we highlight the differences between local time semantics and the standard semantics. We will show that these properties make local time semantics a more apt setting to apply partial order reduction. Finally, we will show that as far as reachability of a state of the network is concerned, local time semantics is equivalent to the standard semantics. In later chapters, we will use local time semantics to develop an alternate, more scalable procedure for the reachability problem of networks of timed automata.

Remark. *In order to clearly distinguish between local time semantics and the standard semantics, we refer to the standard semantics as global semantics. Extending this convention, we sometimes refer to offset valuations, offset zones and offset zone graph as global valuations, global zones and global zone graph, respectively.*

3.1. State-space explosion for networks of timed automata

Recall that we had discussed the challenge posed by state-space explosion to the verification of networks of (untimed) processes in Section 2.10. Observe that in the product automaton of a network of untimed processes, different interleavings of the same sequence lead to the same state. For example, in the transition system T_1 from Figure 2.9, from the state (p_0, q_0, r_0), the sequences abc, bca and cab all lead to the same state (p_2, q_2, r_2). Unfortunately, this is not the case in global zone graphs of networks of timed automata as various timing information, such as the order of reset of clocks, is stored in the zones. We now illustrate this using an example.

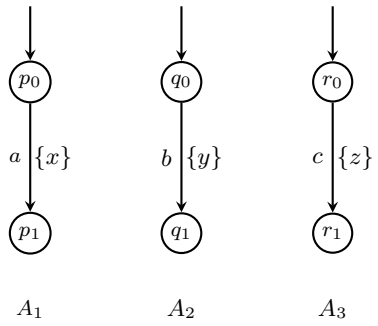


Figure 3.1: Network $A_\times$

Consider the network of timed automata given in Figure 3.1 and the global zone graph of this network given in Figure 3.2. Note that in the network a, b and c are actions local to timed automata A_1, A_2 and A_3

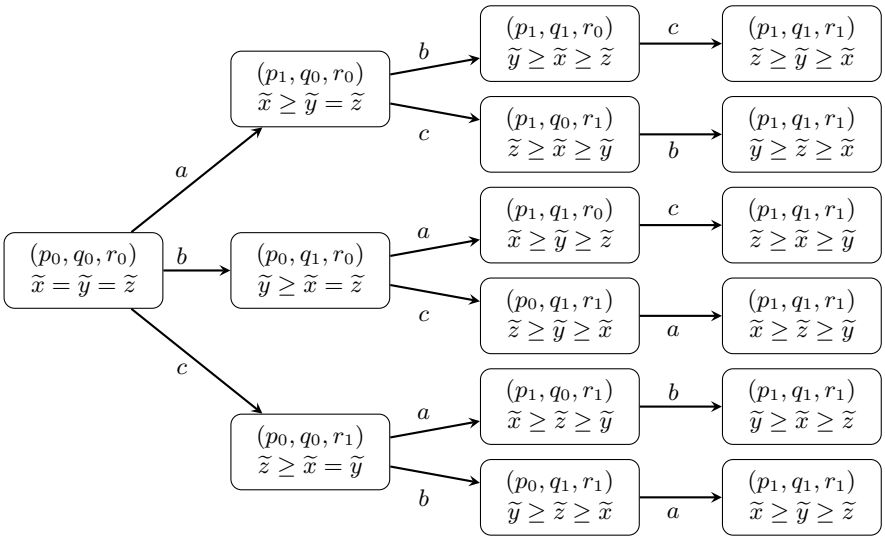


Figure 3.2: Zone graph of $A_\times$ given in Figure 3.1

respectively. Observe that the zone reached on executing an ordering of these actions is different from the zone reached on executing a different ordering. For instance, the run *abc* resets clocks in the order x before y before z, while the run *cab* resets clocks in the order z before x before y. This implies that the valuations of the offset zone reached via the sequence *abc* satisfy the constraint $\widetilde{x} \leq \widetilde{y} \leq \widetilde{z}$, while the valuations of the offset zone reached by the sequence *cab* satisfy the constraint $\widetilde{z} \leq \widetilde{x} \leq \widetilde{y}$. In this way, each execution of a different ordering of actions a, b, c leads to a different zone, as can be observed from Figure 3.2. Note that in an untimed version of this network, we would have only one instance of the state (p_1, q_1, r_1) in the product automaton. (see Figure 1.4.) On the other hand, in the global zone graph from Figure 3.2, we have six nodes with state (p_1, q_1, r_1), one per each possible ordering of clocks $\{x, y, z\}$. This illustrates that the problem of state-space explosion is much more severe for networks of timed automata.

3.2. Why local time semantics?

From the discussion in Section 3.1, it is clear that to have effective verification procedures for large networks of timed automata, we need to have a strategy to combat the blow-up of state space. As we have seen in Section 2.10, partial order reduction is one such technique. An effective way to compute the independence relation between actions is crucial to apply partial order reduction to the transition system.

For networks of untimed systems we have a syntactic check that gives an over-approximation of the independence relation between actions in

the network. Recall that we say that two actions a and b have disjoint domains if the set of processes participating in actions are disjoint, i.e., $\mathsf{dom}(a) \cap \mathsf{dom}(b) = \emptyset$. The following lemma shows that disjointness of domain can be used as an approximation of the independence relation. The proof follows by definitions.

Lemma 3.1. *For every pair of actions a, b of a network of processes, if a and b have disjoint domains, then a and b are independent.*

It is tempting to see if the aforementioned simple syntactic check gives us a sufficient condition for independence of actions in networks of timed automata as well. Unfortunately, it turns out that this check cannot be extended to the networks of timed automata. We illustrate this using the example of a network A which contains two timed automata, A_1 and A_2, as given in Figure 3.3. The action a is a local action of A_1 and b is a local action of A_2. Observe that while the sequence ab is feasible in A, the sequence ba is not feasible. Thus, even if two actions belong to different components, they are not independent in networks of timed automata.

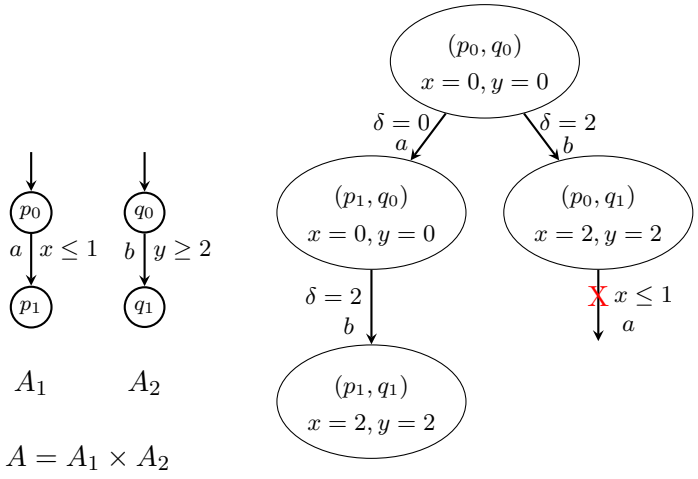


Figure 3.3: Actions with disjoint domains are not independent in A.

The reason for this difference can be interpreted as follows. Recall that an action transition only changes the state and clock values of the components participating in it. Since we do not permit shared clocks in our networks of timed automata, the value of a clock of a component cannot be modified by an action of another component. Thus, in a network of timed automata, the independence relation involving only action transitions is similar to the untimed case, i.e., two action transitions are independent if the set of automata participating in these transitions are disjoint. The independence relation involving only delay transitions is also quite natural. Since a delay

transition changes the value of all the clocks in the network by the same rate, any two delay transitions are mutually independent. However, when we consider the independence relation between an action transition and a delay transition, things get more complicated. Both the action transition and the delay transition could potentially alter the value of a clock of an automaton participating in the action. Therefore, there is a dependency between delay transitions and action transitions.

We will now formalize the ideas that are discussed above. We first state when two executions of a network of timed automata are considered equivalent.

Definition 3.1. We say that two sequences of actions u and w are equivalent, written $u \sim w$, if u can be obtained from w by repeatedly permuting adjacent actions with disjoint domains.

The following lemma shows that if two equivalent sequences are enabled in a network of timed automata, they lead to the same state.

Lemma 3.2. *For two equivalent sequences $u \sim w$: if there are two runs $(q, \overline{v}) \xrightarrow{u} (q_u, \overline{v}_u)$, and $(q, \overline{v}) \xrightarrow{w} (q_w, \overline{v}_w)$, then $q_u = q_w$.*

Proof. Consider the basic case when there are two actions a and b with disjoint domains and $u = ab$, $w = ba$. Since a and b are on disjoint processes, executing ab or ba (whenever possible) leads to the same (discrete) state by definition.

Consider a general u. Sequence w is obtained by repeatedly permuting adjacent actions. From the basic case of the lemma, each permutation preserves the source and target (discrete) states, if it is feasible. □

Observe that in the statement of Lemma 3.2, we cannot assert that $\overline{v}_u = \overline{v}_w$. Even further, the existence of the run $(q, \overline{v}) \stackrel{u}{\Longrightarrow} (q_u, \overline{v}_u)$ does not imply that a run from $(q, \overline{v})$ on w is feasible. As already discussed, this happens due to global time delays, i.e., delays that involve all the processes. For example, consider a network of timed automata A as given in Figure 3.3. The network has actions a having guard $x \leq 1$ and b having guard $y \geq 2$ in process A_1 and A_2, respectively. Observe that from the initial valuation one can execute ab but not ba. This illustrates that, unlike for untimed networks, disjointness of domain does not imply independence in networks of timed automata.

To summarize, since the elapse of time in each process is synchronized at all points of time, there are implicit dependencies between actions of different processes, induced by the global nature of time. If the elapse of time is decoupled, then we may be able to recover commutativity between actions with disjoint domains.

We consider *local time semantics* as an alternate semantics for networks of timed automata that satisfies this requirement. Introduced by Bengtsson

et al. in 1998 [BJLY98], in this semantics, time elapses independently in each process. Whenever a set of processes execute a synchronized action (see Definition 2.8) the reference times of all these processes are synchronized. This way, two actions with disjoint domains become commutative.

Consider the example of the network from Figure 3.3. Recall that the run ba was not feasible in the standard semantics of the network. Observe that in local time semantics, while the process A_2 executing b elapses 2 time units, A_1 is allowed to not elapse time at all. As a consequence, the run ba becomes feasible. The local runs are as shown in Figure 3.4.

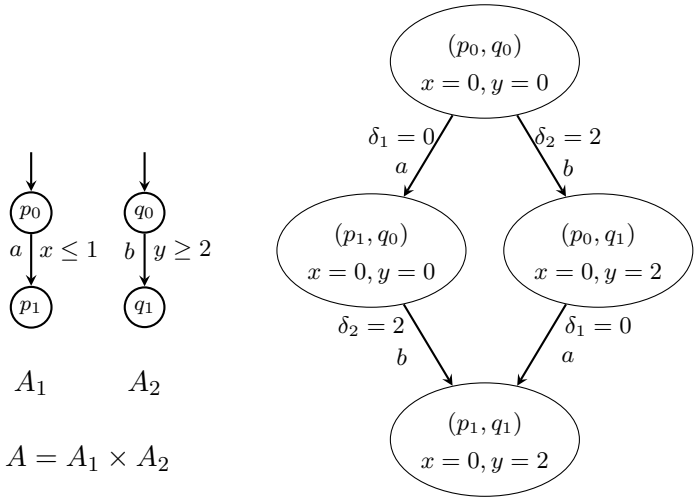


Figure 3.4: Local runs of the network A.

3.3. Local valuations

Recall the definition of offset valuations in Definition 2.10. As in the case of offset valuations, we choose an offset representation for local valuations, the reasons for which we will discuss later. Thus, for each clock x, we have an offset variable $\widetilde{x}$ that stores the time-stamp at which x was last reset. We replace the reference clock t which was tracking the global time, with individual *reference clocks* t_p for each process A_p which track the local time of each process. We set $\widetilde{X}'_p = \widetilde{X}_p \cup \{t_p\}$ and $\widetilde{X}' = \bigcup_p \widetilde{X}'_p$.

Definition 3.2. A *local valuation* v is a valuation over the set of clocks $\widetilde{X}'$ such that $\mathsf{v}(\widetilde{x}) \leq \mathsf{v}(t_p)$ for all processes $p \in \mathsf{Proc}$ and all clocks $\widetilde{x} \in \widetilde{X}_p$.

This restriction captures the intuition that t_p is a reference clock for process p, and it is never reset. In this setting, the value $\mathsf{v}(t_p) - \mathsf{v}(\widetilde{x})$ of clock

x is defined relative to the reference clock t_p of process p such that $x \in X_p$. We will use the notation v for local valuations to distinguish from standard valuations v and offset valuations $\overline{v}$. We will sometimes write $\mathsf{v}(x-y)$ to denote $\mathsf{v}(x) - \mathsf{v}(y)$.

We formally define the operations in local time semantics below.

- **Local-time elapse**: For a process $p \in \mathsf{Proc}$ and $\delta \in \mathbb{R}_{\geq 0}$, the operation $\mathsf{v} +_p \delta$ increments $\mathsf{v}(t_p)$, the value of the reference clock t_p of process p by δ, and leaves all the other variables unchanged. Formally,

$$(\mathsf{v} +_p \delta)(t_p) = \mathsf{v}(t_p) + \delta$$
$$(\mathsf{v} +_p \delta)(x) = \mathsf{v}(x) \quad \text{for all} \quad x \in \widetilde{X}' \setminus \{t_p\}$$

- **Reset**: We denote by $[R]\mathsf{v}$ the valuation obtained after resetting the clocks in $R \subseteq X$ and defined by:

$$\begin{aligned} ([R]\mathsf{v})(t_p) &= \mathsf{v}(t_p) \quad \text{for all reference clocks } t_p \\ ([R]\mathsf{v})(\widetilde{x}) &= \mathsf{v}(t_p) \quad \text{if } x \in R \text{ and } x \in X_p \\ &= \mathsf{v}(\widetilde{x}) \quad \text{otherwise} \end{aligned}$$

  Consider a local valuation v of a network of two timed automata A_1 and A_2, such that x is the clock of A_1 and y is the clock of A_2.

$$\mathsf{v} : t_1 = 4, \widetilde{x} = 0, t_2 = 4, \widetilde{y} = 2$$

  The valuation v' obtained on reset of clock x is as given below.

$$\mathsf{v}' : t_1 = 4, \widetilde{x} = 4, t_2 = 4, \widetilde{y} = 2$$

- **Guard satisfaction**: A local valuation v satisfies a clock constraint g, denoted $\mathsf{v} \models g$ if each constraint in g holds upon replacing x by value $\mathsf{v}(t_p) - \mathsf{v}(\widetilde{x})$ where p is the process such that $x \in X_p$.

Next, we define the notion of a *synchronized local valuation*, sometimes simply referred to as synchronized valuation.

Definition 3.3. A local valuation v is *synchronized* if for every pair of processes p_1, p_2, the values of their reference clocks are equal: $\mathsf{v}(t_{p_1}) = \mathsf{v}(t_{p_2})$.

For a synchronized local valuation v, let $\mathsf{global}(\mathsf{v})$ be the offset valuation $\overline{v}$ such that $\overline{v}(\widetilde{x}) = \mathsf{v}(\widetilde{x})$ for all offset clocks $\widetilde{x} \in \widetilde{X}$ and $\overline{v}(t) = \mathsf{v}(t_1) = \cdots = \mathsf{v}(t_k)$. Conversely, to every offset valuation $\overline{v}$, we associate the synchronized local valuation $\mathsf{local}(\overline{v}) = \mathsf{v}$ where $\mathsf{v}(\widetilde{x}) = \overline{v}(\widetilde{x})$ and $\mathsf{v}(t_p) = \overline{v}(t)$ for every reference clock t_p.

We now state Lemma 3.3 that relates the operations on synchronized local valuations to operations on offset valuations.

Lemma 3.3. *Suppose* v *is a synchronized local valuation and* $\overline{v} = \mathsf{global}(\mathsf{v})$. *Then, we have the following:*

- $\overline{v} \models g$ *iff* $\mathsf{v} \models g$.
- *Let* $\overline{v}' = [R]\overline{v}$ *and* $\mathsf{v}' = [R](\mathsf{v})$. *Then,* $\overline{v}' = \mathsf{global}(\mathsf{v}')$.
- *Let* $\overline{v}' = \overline{v} + \delta$, *and* $\mathsf{v}' = \mathsf{v} +_p \delta$, *for all processes* $p \in Proc$. *Then,* $\overline{v}' = \mathsf{global}(\mathsf{v}')$.

  Proof. The first item follows from the definitions of guard satisfaction for local valuations and offset valuations.

  For the second item, we know that $\overline{v}'(t) = \overline{v}(t)$, and $\overline{v}'(\widetilde{x}) = \overline{v}(\widetilde{x})$ if $x \notin R$ and $\overline{v}'(\widetilde{x}) = \overline{v}'(t)$ otherwise. We also have $\mathsf{v}'(t_p) = \mathsf{v}(t_p)$, for all reference clocks t_p and $\mathsf{v}'(\widetilde{x}) = \mathsf{v}(\widetilde{x})$, if $x \notin R$ and $\mathsf{v}'(\widetilde{x}) = \mathsf{v}'(t_p)$, where $x \in X_p$, otherwise. Then, we observe that if $x \notin R$, $\mathsf{v}'(\widetilde{x}) = \mathsf{v}(\widetilde{x}) = \overline{v}'(\widetilde{x})$. If $x \in R$, $\mathsf{v}'(x) = \mathsf{v}'(t_p) = \mathsf{v}(t_p) = \overline{v}(t) = \overline{v}'(t)$. Thus, we have $\overline{v}' = \mathsf{global}(\mathsf{v}')$.

  For the third item, we know that $\overline{v}'(t) = \overline{v}(t) + \delta$, and $\overline{v}'(\widetilde{x}) = \overline{v}(\widetilde{x})$ for all offset clocks $\widetilde{x}$. About v', we know $\mathsf{v}'(t_p) = \mathsf{v}(t_p) + \delta$, for all reference clocks t_p. Since v was synchronized, we know that v' is too, and since $\overline{v} = \mathsf{global}(\mathsf{v})$, we have $\mathsf{global}(\mathsf{v}')(t) = \overline{v}(t) + \delta$. Further, $\mathsf{v}'(\widetilde{x}) = \mathsf{v}(\widetilde{x}) = \overline{v}(\widetilde{x}) = \overline{v}'(\widetilde{x})$ for all offset clocks $\widetilde{x}$. Thus, we have $\overline{v}' = \mathsf{global}(\mathsf{v}')$. □

We now define the local step of a network of timed automata.

Definition 3.4 (Local steps of a network of timed automata)**.** There are two kinds of local steps in a network $\mathcal{N}$: *local delay*, and *local action*. A local delay $\delta \in \mathbb{R}_{\geq 0}$ in process $p \in Proc$ is a step $(q, \mathsf{v}) \xrightarrow{p,\delta}_{st} (q, \mathsf{v} +_p \delta)$. For an action b, we have a step $(q, \mathsf{v}) \xrightarrow{b}_{st} (q', \mathsf{v}')$ if for each $p \in \mathsf{dom}(b)$ the unique b transition of p is $T_p(b) = (q(p), g_p, R_p, q'(p))$, and the following hold:

- start times are synchronized: $\mathsf{v}(t_{p_1}) = \mathsf{v}(t_{p_2})$, for each $p_1, p_2 \in \mathsf{dom}(b)$;
- guards are satisfied: $\mathsf{v} \vDash g_p$, for each $p \in \mathsf{dom}(b)$;
- resets are performed: $\mathsf{v}' = [\bigcup_{p \in \mathsf{dom}(b)} R_p]\mathsf{v}$;
- other processes do not move: $q(p) = q'(p)$, for each $p \notin \mathsf{dom}(b)$.

The main difference between local time semantics and global semantics is the presence of local delay. As a result, each process can be in a different local time as emphasized by the reference clocks in each process. In consequence, in local action steps we require that, when processes execute a common action, their local times should be the same. We will now give some examples

of local steps. Consider the network A given in Figure 3.5. Consider the following run σ of A,

$$(s_0, r_0, \mathsf{v}_0) \xrightarrow{p_1,\delta_1} (s_0, r_0, \mathsf{v}_1) \xrightarrow{a} (s_0, r_1, \mathsf{v}_2)$$

In this run, $(s_0, r_0, \mathsf{v}_0) \xrightarrow{p_1,\delta_1} (s_0, r_0, \mathsf{v}_1)$ is a local delay step and $(s_0, r_0, \mathsf{v}_1) \xrightarrow{a} (s_0, r_1, \mathsf{v}_2)$ is a local action step.

Observe that a standard delay δ on all processes can be simulated by a sequence of local delays on every process separately, as $\xrightarrow{1,\delta}_{st} \cdots \xrightarrow{k,\delta}_{st}$. For a sequence of local delays $\Delta = (p_1, \delta_1) \dots (p_n, \delta_n)$ we will write $(q, \mathsf{v}) \xrightarrow{\Delta}_{st} (q, \mathsf{v}')$ to mean $(q, \mathsf{v}) \xrightarrow{p_1,\delta_1}_{st} (q, \mathsf{v}_1) \xrightarrow{p_2,\delta_2}_{st} \cdots \xrightarrow{(p_n,\delta_n)}_{st} (q, \mathsf{v}')$.

Definition 3.5 (Local run)**.** A *local run* from a configuration (q_0, v_0) is a sequence of local steps. For a sequence of actions $u = b_1 \dots b_n$, we write $(q_0, \mathsf{v}_0) \xrightarrow{u} (q_n, \mathsf{v}'_n)$ if for some sequences of local delays $\Delta_0, \dots, \Delta_n$ there is a local run

$$(q_0, \mathsf{v}_0) \xrightarrow{\Delta_0}_{st} (q_0, \mathsf{v}'_0) \xrightarrow{b_1}_{st} (q_1, \mathsf{v}_1) \xrightarrow{\Delta_1}_{st} \cdots \xrightarrow{b_n}_{st} (q_n, \mathsf{v}_n) \xrightarrow{\Delta_n}_{st} (q_n, \mathsf{v}'_n)$$

Observe that a run may start and end with a sequence of delays.

3.4. Independence in local time semantics

In this section, we will discuss some properties of runs in local time semantics. We will compare and contrast local runs and global runs, and show that local runs have a key independence property that is not true for global runs. First, we will show that, in contrast to global semantics, in local time semantics, two actions with disjoint domains satisfy the conditions for independence between actions.

Lemma 3.4 (Independence)**.** *Suppose that* $\mathsf{dom}(a) \cap \mathsf{dom}(b) = \emptyset$*. If* $(q, \mathsf{v}) \xrightarrow{ab} (q', \mathsf{v}')$*, then* $(q, \mathsf{v}) \xrightarrow{ba} (q', \mathsf{v}')$*. If* $(q, \mathsf{v}) \xrightarrow{a} (q_a, \mathsf{v}_a)$ *and* $(q, \mathsf{v}) \xrightarrow{b} (q_b, \mathsf{v}_b)$*, then* $(q, \mathsf{v}) \xrightarrow{ab} (q_{ab}, \mathsf{v}_{ab})$ *for some* q_{ab} *and* v_{ab}*.*

Proof. Take a run $(q, \mathsf{v}) \xrightarrow{\Delta_a} (q, \mathsf{v}_a) \xrightarrow{a} (q_a, \mathsf{v}'_a) \xrightarrow{\Delta_b} (q_a, \mathsf{v}_b) \xrightarrow{b} (q', \mathsf{v}'_b) \xrightarrow{\Delta} (q', \mathsf{v}')$. Let Δ'_b be the sequence of delays from Δ_a or Δ_b involving processes in $\mathsf{dom}(b)$, i.e., pairs (p, δ_p) from Δ_a or Δ_b such that $p \in \mathsf{dom}(b)$. Let Δ'_a be a sequence of delays in Δ_a involving processes in $\mathsf{dom}(a)$; and finally let Δ' be the delays in $\Delta_a \cup \Delta_b \cup \Delta$ which were not counted in Δ'_a or Δ'_b. Since $\mathsf{dom}(a) \cap \mathsf{dom}(b) = \emptyset$ we get that the following sequence is a run: $(q, \mathsf{v}) \xrightarrow{\Delta'_b} (q, \mathsf{v}''_b) \xrightarrow{b} (q_b, \mathsf{v}'''_b) \xrightarrow{\Delta'_a} (q', \mathsf{v}''_a) \xrightarrow{a} (q', \mathsf{v}'''_a) \xrightarrow{\Delta'} (q', \mathsf{v}')$.

For the second part, suppose that we have the sequences $(q, \mathsf{v}) \xrightarrow{\Delta_a} \xrightarrow{a} \xrightarrow{\Delta'_a} (q_a, \mathsf{v}_a)$ and $(q, \mathsf{v}) \xrightarrow{\Delta_b} \xrightarrow{b} \xrightarrow{\Delta'_b} (q_b, \mathsf{v}_b)$. Let Δ_1 be the delays involving processes in $\mathsf{dom}(a)$ in Δ_a, and Δ_2 the delays of processes in $\mathsf{dom}(b)$ in Δ_b. As $\mathsf{dom}(a) \cap \mathsf{dom}(b) = \emptyset$, $(q, \mathsf{v}) \xrightarrow{\Delta_1} \xrightarrow{a} \xrightarrow{\Delta_2} \xrightarrow{b} (q_{ab}, \mathsf{v}_{ab})$ is a local run. □

We illustrate the independence of actions with disjoint domains using an example, given in Figure 3.5. Observe that the network is the same as that in Figure 3.3. From the runs shown, it can be seen that in local time semantics, both ab and ba are feasible. This is in contrast with the global semantics where the sequence ba is not feasible.

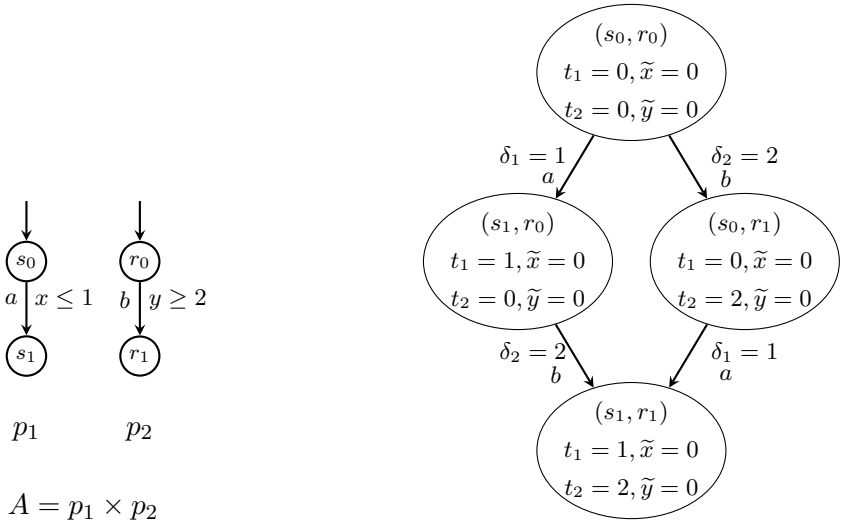


Figure 3.5: Independence in local time semantics

Recall that two sequences of actions are equivalent, written $u \sim w$ if one can be obtained from the other by repeatedly permuting adjacent actions with disjoint domains (see Definition 3.1). Directly from Lemma 3.4, we obtain

Lemma 3.5. *If* $(q_0, \mathsf{v}_0) \xrightarrow{u} (q_n, \mathsf{v}_n)$ *and* $u \sim w$, *then* $(q_0, \mathsf{v}_0) \xrightarrow{w} (q_n, \mathsf{v}_n)$.

Proof. Since $u \sim w$, we know that w can be obtained from u by repeatedly permuting adjacent actions with disjoint domains.

Let the local run u be of the form

$$(q_0, \mathsf{v}_0) \xrightarrow{b_1} (q_1, \mathsf{v}_1) \xrightarrow{b_2} \cdots (q_{i-1}, \mathsf{v}_{i-1}) \xrightarrow{b_i} (q_i, \mathsf{v}_i) \xrightarrow{b_{i+1}} (q_{i+1}, \mathsf{v}_{i+1}) \cdots \xrightarrow{b_n} (q_n, \mathsf{v}'_n)$$

Suppose that b_{i+1} is the first action of w. We know from the definition of $\sim$ that for $0 \leq k \leq i$, b_k and b_{i+1} have disjoint domains. By repeatedly applying Lemma 3.4, we can commute b_{i+1} with b_i, b_{i-1} and so on up to b_1. After doing this, we obtain a local run of the following form:

$$(q_0, \mathsf{v}_0) \xrightarrow{b_{i+1}} (q'_1, \mathsf{v}'_1) \xrightarrow{b_1} \cdots (q'_{i-1}, \mathsf{v}'_{i-1}) \xrightarrow{b_{i-1}} (q'_i, \mathsf{v}'_i) \xrightarrow{b_i} (q_{i+1}, \mathsf{v}_{i+1}) \cdots \xrightarrow{b_n} (q_n, \mathsf{v}'_n)$$

Since $u \sim b_{i+1}u'$ and $w = b_{i+1}w'$, we can use induction for the pair $u' \sim w'$ to obtain a run on w' from (q'_1, v'_1) to (q_n, v_n). □

Thus, in local time semantics, if a run u exists from (q_0, v_0) and $u \sim w$, then w is guaranteed to exist. Recall that w need not be feasible in global semantics. Further, the two equivalent sequences not only reach the same state q_n, but also the same local valuation v_n (again in contrast with Lemma 3.2 for global-time semantics).

3.5. Equivalence of local and global runs

We show below that, despite local runs having much more freedom than global runs (as time can elapse independently in every process), with respect to state reachability, the two concepts turn out to be equivalent.

Remark. *In order to distinguish between steps in global semantics and local time semantics, we denote transitions in local time semantics by $\longrightarrow$, while transitions in global semantics are denoted by $\Longrightarrow$.*

Before we prove the equivalence between global and local runs, we develop some intermediate observations.

Every action step in a local run:

$$(q_0, \mathsf{v}_0) \xrightarrow{\Delta_0}_{st} (q_0, \mathsf{v}'_0) \xrightarrow{b_1}_{st} (q_1, \mathsf{v}_1) \ldots \xrightarrow{\Delta_{n-1}}_{st} (q_{n-1}, \mathsf{v}'_{n-1}) \xrightarrow{b_n}_{st} (q_n, \mathsf{v}_n) \xrightarrow{\Delta_n}_{st} (q_n, \mathsf{v}'_n)$$

has its execution time; namely the step $(q_{i-1}, \mathsf{v}'_{i-1}) \xrightarrow{b_i}_{st} (q_i, \mathsf{v}_i)$ has the execution time $\mathsf{v}'_{i-1}(t_p) = \mathsf{v}_i(t_p)$ for $p \in \mathsf{dom}(b_i)$. Observe that by definition of a step, the choice of p does not matter, as long as p is in the domain of b_i.

We will say that a local run is *soon* if for every i, the execution time of b_i is not bigger than the execution time of b_{i+1}.

Lemma 3.6. *If $(q_0, \mathsf{v}_0) \xrightarrow{u} (q_n, \mathsf{v}_n)$ is a local run, then there is $w \sim u$ such that $(q_0, \mathsf{v}_0) \xrightarrow{w} (q_n, \mathsf{v}_n)$ is a soon local run.*

Proof. Consider a sequence $u = b_1 \ldots b_n$ and a run $(q_0, \mathsf{v}_0) \xrightarrow{\Delta_0}_{st} (q_0, \mathsf{v}'_0) \xrightarrow{b_1}_{st} (q_1, \mathsf{v}_1) \cdots \xrightarrow{\Delta_{n-1}}_{st} (q_{n-1}, \mathsf{v}'_{n-1}) \xrightarrow{b_n}_{st} (q_n, \mathsf{v}_n) \xrightarrow{\Delta_n}_{st} (q_n, \mathsf{v}'_n)$. Suppose that the order of execution times of b_i and b_{i+1} is reversed. Then, $\mathsf{dom}(b_i) \cap$

$\mathsf{dom}(b_{i+1}) = \emptyset$ by the definition of a run. So we can take $u' = b_1 \dots b_{i+1} b_i \dots b_n$ where the order of b_i and b_{i+1} is reversed. Since $u \sim u'$, by Lemma 3.5 we have a run $(q_0, \mathsf{v}_0) \xrightarrow{u'} (q_n, \mathsf{v}_n)$. We can, if necessary, repeat this operation from u' till we get the desired w. □

Lemma 3.7. *If* $(q, \mathsf{v}) \xrightarrow{u} (q', \mathsf{v}')$ *is a local run where* v *and* v' *are synchronized valuations, then there is* $w \sim u$ *and a global run* $(q, \mathsf{global}(\mathsf{v})) \overset{w}{\Longrightarrow} (q', \mathsf{global}(\mathsf{v}'))$.

Proof. Let $(q, \mathsf{v}) = (q_0, \mathsf{v}_0)$ and $(q', \mathsf{v}') = (q_n, \mathsf{v}'_n)$. We take a local run, and assume that it is soon thanks to Lemma 3.6:

$$(q_0, \mathsf{v}_0) \xrightarrow{\Delta_0}_{st} (q_0, \mathsf{v}'_0) \xrightarrow{b_1}_{st} (q_1, \mathsf{v}_1) \xrightarrow{\Delta_1}_{st} \cdots$$
$$\cdots (q_{n-1}, \mathsf{v}_{n-1}) \xrightarrow{\Delta_{n-1}}_{st} (q_{n-1}, \mathsf{v}'_{n-1}) \xrightarrow{b_n}_{st} (q_n, \mathsf{v}_n) \xrightarrow{\Delta_n}_{st} (q_n, \mathsf{v}'_n) \ .$$

Let θ_i be the execution time of action b_i. For convenience, we set $\theta_0 = \mathsf{v}_0(t_p)$ and $\theta_{n+1} = \mathsf{v}'_n(t_p)$, for some process p. Since v_0 and v'_n are synchronized, the choice of p is irrelevant. We claim that there is a global run

$$(q_0, \overline{v}_0) \overset{\delta_1}{\Longrightarrow}_{st} \overset{b_1}{\Longrightarrow}_{st} (q_1, \overline{v}_1) \overset{\delta_2}{\Longrightarrow}_{st} \overset{b_2}{\Longrightarrow}_{st} \cdots \overset{\delta_i}{\Longrightarrow}_{st} \overset{b_i}{\Longrightarrow}_{st} (q_i, \overline{v}_i)$$

with

- $\overline{v}_0 = \mathsf{global}(\mathsf{v}_0)$
- $\delta_i = \theta_i - \theta_{i-1}$ for $i = 1, \dots, n$
- $\overline{v}_i(t) = \mathsf{v}_i(t_p)$ for some $p \in \mathsf{dom}(b_i)$ and
- $\overline{v}_i(\widetilde{x}) = \mathsf{v}_i(\widetilde{x})$ for all other clocks x

This statement is proved by induction on i. For $i = n$, this statement gives an offset valuation $\overline{v}_n$ such that $\overline{v}_n(t) = \mathsf{v}_n(t_p)$ where $p \in \mathsf{dom}(b_n)$ and for all other clocks $\overline{v}_n(\widetilde{x}) = \mathsf{v}_n(\widetilde{x})$. Note that valuations v_n and v'_n differ only in the values of the reference clocks. Moreover, in v'_n, all reference clocks are at θ_{n+1}. A global delay of $\delta_{n+1} = \theta_{n+1} - \theta_n$ from $(q_n, \overline{v}_n)$ gives $(q_n, \overline{v}'_n)$ such that $\overline{v}'_n = \mathsf{global}(\mathsf{v}'_n)$. □

Lemma 3.8. *If* $(q, \overline{v}) \overset{u}{\Longrightarrow} (q', \overline{v}')$ *is a global run, then there is a local run* $(q, \mathsf{local}(v)) \xrightarrow{u} (q', \mathsf{local}(v'))$.

Proof. A global run can be directly converted to a local run by changing a global delay to a sequence of local delays. □

From Lemma 3.7 and Lemma 3.8, we get the following lemma.

Lemma 3.9. *If* $(q, \mathsf{v}) \xrightarrow{u} (q', \mathsf{v}')$ *is a local run where* v *and* v' *are synchronized local valuations, there exists a global run* $(q, \mathsf{global}(\mathsf{v})) \xRightarrow{w} (q', \mathsf{global}(\mathsf{v}'))$ *for some* $w \sim u$*. Conversely, if* $(q, \overline{v}) \xRightarrow{u} (q', \overline{v}')$ *is a global run, then there is a local run* $(q, \mathsf{local}(v)) \xrightarrow{u} (q', \mathsf{local}(v'))$*.*

The reachability problem with respect to local time semantics is defined as before: q is reachable if there is a local run $(q_0, \mathsf{v}_0) \xrightarrow{u} (q, \mathsf{v})$ for some v where $\mathsf{v}_0 = \mathsf{local}(\overline{v}_0)$. Note that here, q_0 and v_0 denote the initial state and the initial (local) valuation respectively. By adding some local delays at the end of the run we can always assume that v is synchronized. Lemma 3.9 thus implies that the reachability problem in local time semantics is equivalent to the standard one in global semantics.

Chapter 4

Local zone graph

In Chapter 3, we introduced the local time semantics for networks of timed automata. The goal of this chapter is to introduce concepts analogous to global zones (Definition 2.30) and global zone graphs (Definition 2.34) for the setting of local time semantics.

We first show that it is not possible to represent sets of valuations obtained via runs in the local time semantics using the standard representation of zones. This is because the standard representation of zones is not compatible with the operations of local time semantics (specifically, the local time elapse operation) as illustrated in Lemma 4.1. This means that we need a different representation of zones to represent sets of local valuations. To this end, we introduce the notion of *local zones*, which uses difference constraints involving offset clocks and reference clocks. We show that local zones are closed under the operations of local time semantics.

We then introduce the concept of *local zone graph* as the analogue of the global zone graph for the setting of local time semantics. Two sequences are said to be equivalent if one can be obtained from the other by commuting adjacent actions with disjoint domains. We show that in the local zone graph, actions that have disjoint domains satisfy the diamond property (see Definition 2.48.) Extending this observation, we show that local zone graphs satisfy the following very useful property: if a sequence of actions is feasible from a node of the local zone graph, then all sequences equivalent to this sequence are feasible from that node, and moreover, they all lead to the same node. This suggests that as in the case of untimed networks, in local zone graphs, disjoint domains can be used as a condition for independence between actions. Since our goal is to apply partial-order reduction to networks of timed automata, the aforementioned property gives us incentive to use local zone graphs over global zone graphs to answer the reachability problem. However, we point out that the local zone graph may be infinite and so cannot be used directly to check reachability of networks of timed automata.

The results in this chapter appeared in [GHSW19].

4.1. Why local zones?

The goal of this section is to introduce local zones, a concept similar to zones, but for local time semantics. The specificity of local time semantics is local time elapse. So a natural first attempt to do this is to use the representation which was used for standard zones (see Section 2.5), and implement local time elapse as an operation on zones. A zone in such a standard representation is defined by constraints $y_1 - y_2 \lessdot c$ where $c \in \mathbb{Z}$ and $y_1, y_2 \in X \cup \{t_1, \dots t_k\}$. A prerequisite for being able to use a representation of zones for the symbolic reachability checking algorithm based on local time semantics is that it should be closed under the operations of local time semantics that were discussed in Chapter 3, namely guard intersection, reset and local time elapse. In Lemma 4.1, we show that the standard representation of zones is not closed under local time elapse operation.

Lemma 4.1. *In the standard representation of sets of local valuations, there is a zone Z such that the set of all valuations reachable by local time elapse from Z is not a zone.*

Proof. Consider a set of clocks $X = \{x_1, t_1, x_2, t_2\}$, and a zone Z defined by the following constraints:

$$t_1 = t_2, \qquad x_1 = x_2, \qquad x_1 \leq t_1, \qquad x_2 \leq t_2$$

The last two constraints ensure that Z contains only local valuations. Note that there is a network of timed automata where Z can indeed be reached from the initial valuation.

Consider the set D of local valuations obtained after a local delay from some valuation in Z. It is easy to see that each valuation in D satisfies the constraint $t_1 - x_1 = t_2 - x_2$ since Z satisfies this constraint, and the constraint is invariant under local time elapse. We cannot use this constraint directly to define a zone, since it is not of the form $y_1 - y_2 \lessdot c$.

Suppose that D is a zone. We consider the constraints of the form $y_1 - y_2 \lessdot c$ that are satisfied in D.

- No constraint of the form $y - 0 \lessdot c$ can be true for all valuations in D, since Z contains valuations with arbitrary big values for each variable $y \in X$.
- A constraint of the form $0 - y \lessdot c$ can be true only if $c \geq 0$. Indeed, this constraint translates to $-c \lessdot y$, and D could have a valuation with arbitrary small values of y. As a consequence, D satisfies a trivial constraint $y \geq 0$.
- Constraints $t_i - x_i \lessdot c$ cannot hold in D, as the value of t_i can be arbitrarily bigger than the value of x_i.

- A constraint $x_i - t_i \lessdot c$ holds in D only if $c \geq 0$, since the difference between x_i and t_i can be arbitrarily small in Z.
- All the remaining constraints are of the form $y_j - y_i \lessdot c$ with $y_i \in \{x_i, t_i\}$, $y_j \in \{x_j, t_j\}$, and $i \neq j$. None of them can hold because local time elapse can make this difference arbitrarily big.

This implies that the only zone constraints satisfied by D are $x_i \leq t_i$. However, these constraints do not imply $t_1 - x_1 = t_2 - x_2$. As a consequence, D cannot be a zone. □

Thus, it is not possible to use the standard representation of zones to represent local time transitions.

4.2. Local zones

In this section, we will introduce *local zones* to represent sets of local valuations. As demonstrated using Lemma 4.1, the standard method of storing difference constraints between clocks is not sufficient to capture all the operations of local time semantics. So, we will adopt an offset representation of clock valuations to store sets of local valuations, where for each clock x, we have an offset clock $\widetilde{x}$ that stores the time-stamp at which x was last reset. Local zones are defined by difference constraints involving offset clocks and reference clocks.

Remark. *Local zones can be viewed as the offset representation of standard zones extended to the setting of local time semantics. The main difference is that since the progress of time is synchronized in global semantics, offset zones uses only one reference clock t. Since the progress of time is decoupled in local time semantics local zones use a reference clock t_p for each process A_p in the network.*

We now formally define local zones. Recall that $\widetilde{X}_p = \{\widetilde{x} \mid x \in X_p\}$, $\widetilde{X}'_p = \widetilde{X}_p \cup \{t_p\}$ and $\widetilde{X}' = \bigcup_p \widetilde{X}'_p$.

Definition 4.1. A *local zone* is a zone over local valuations: a set of local valuations defined by constraints $y_1 - y_2 \lessdot c$ where $y_1, y_2 \in \widetilde{X}'$.

Recall that a local valuation v satisfies $\mathsf{v}(\widetilde{x}) \leq \mathsf{v}(t_p)$ for each process p and each $\widetilde{x} \in \widetilde{X}_p$. We denote local zones using the notation Z along with subscripts or superscripts, to distinguish from standard zones and offset zones. We say that a local valuation v belongs to Z, written as $\mathsf{v} \in \mathsf{Z}$, if $\mathsf{v}(y_1) - \mathsf{v}(y_2) \lessdot c$ for each constraint $y_1 - y_2 \lessdot c$ of Z. We say that a zone Z is said to be in canonical form if each constraint defining Z is tight.

We say that two local valuations v_1 and v_2 are same up-to time shift, denoted as $\mathsf{v}_1 \sim_{ts} \mathsf{v}_2$, if $\mathsf{v}_1(x-y) = \mathsf{v}_2(x-y)$ for all $x, y \in \widetilde{X}'$. The following lemma establishes that local zones are closed under time shift.

Lemma 4.2. *For each local zone* Z*, and local valuations* $\mathsf{v}_1 \sim_{ts} \mathsf{v}_2$ *which are same up-to time shift,* $\mathsf{v}_1 \in \mathsf{Z}$ *iff* $\mathsf{v}_2 \in \mathsf{Z}$*.*

Proof. Since $\mathsf{v}_1(x-y) = \mathsf{v}_2(x-y)$ for all pairs of clocks, valuation v_1 satisfies a constraint $x - y \lessdot c$ iff v_2 satisfies it. □

Recall the distance graph representation for offset zones discussed in Definition 2.32. We have similar notions of distance graph for local zones also. Recall that a distance graph is said to be in canonical form if for each pair of variables y_1, y_2, the shortest path from y_2 to y_1 is given by the weight of the edge $y_2 \to y_1$. Sometimes, we use the term *removing* an edge $y_2 \to y_1$ from G to mean that the value of this edge is assigned as $(<, \infty)$. Given two distance graphs G_1, G_2, we write $\min(G_1, G_2)$ for the graph where each edge weight is the minimum of the corresponding weights from G_1 and G_2. For a distance graph G, we write $[\![G]\!]$ for the set of solutions to the constraints given by G. We have $[\![\min(G_1, G_2)]\!] = [\![G_1]\!] \cap [\![G_2]\!]$.

Operations on local zones

We now define the following operations on local zones, corresponding to the basic operations of local time semantics namely local time elapse, intersection with a guard, and reset of clocks:

- local-elapse$(\mathsf{Z}) = \{v +_p \delta_p : v \in \mathsf{Z}, \delta_p \in \mathbb{R}_{\geq 0}\}$ where delay is applied for each process p,
- $\mathsf{Z}_g = \{\mathsf{v} \mid \mathsf{v} \vDash g\}$,
- $[R]\mathsf{Z} = \{[R]\mathsf{v} \mid \mathsf{v} \in \mathsf{Z}\}$ for every $R \subseteq X$.

We will show that local zones are closed under all these operations.

Lemma 4.3. *For a local zone* Z*, the set* local-elapse(Z) *is a local zone.*

Proof. We define the following operation on distance graphs: local-elapse(G) is a distance graph obtained from distance graph G by removing all edges $y \to t_p$ where $y \in \widetilde{X}'$ and t_p is a reference clock. Note that these edges are the incoming edges to reference clocks.

Let G_{Z} be the canonical distance graph of Z. We will prove that the distance graph local-elapse(G_{Z}) is the canonical distance graph representing local-elapse(Z).

For notational convenience we will consider an operation of time elapse for one process, say A_p. Then, the time-elapse operation local-elapse is a composition of time elapse for all processes A_p.

We consider the set

$$\mathsf{Z}^+ = \{v +_p \delta_p : v \in \mathsf{Z}, \delta_p \in \mathbb{R}_{\geq 0}\}$$

and the distance graph G^+ obtained from G_Z by removing all incoming edges to t_p.

First observe that G^+ is canonical. This means that the value of each edge $x \to y$ in G^+ is the minimum over the paths from x to y. If y is not a reference clock, then this is straightforward, as all paths from x to y in G^+ exist also in G_Z and G_Z is canonical. If y is a reference clock (say t_p), then there are no paths arriving at t_p, so indeed the value of the shortest path from x to t_p is ∞.

The inclusion $\mathsf{Z}^+ \subseteq [\![G^+]\!]$ is direct. Each valuation in Z^+ satisfies all the constraints given by the edges of G_Z except for the edges $x \to t_p$.

It remains to prove $[\![G^+]\!] \subseteq \mathsf{Z}^+$. Consider a v^+ in Z^+. Consider two quantities:

$$d_1 = \max\{\mathsf{v}^+(t_p) - \mathsf{v}^+(x) - c_{xt_p} : x \xrightarrow{\leq c_{xt_p}} y \text{ is a edge in } G\}$$
$$d_2 = \max\{\mathsf{v}^+(t_p) - \mathsf{v}^+(x) - c_{xt_p} : x \xrightarrow{< c_{xt_p}} y \text{ is a edge in } G\}$$

Notice that quantity d_1 is considered over edges $x \xrightarrow{\leq c_{xt_p}} y$ with a weak inequality and d_2 over edges $x \xrightarrow{< c_{xt_p}} y$ with a strict inequality.

If $d_1 \leq 0$ and $d_2 < 0$, then v^+ satisfies all the constraints of G_Z, so $\mathsf{v}^+ \in \mathsf{Z}$. Otherwise, either $d_1 = \max(d_1, d_2)$ and $d_1 > 0$ or $d_2 = \max(d_1, d_2)$ and $d_2 \geq 0$. In the former case, set $d := d_1$. In the latter case, set $d := d_2 + \varepsilon$ where $\varepsilon > 0$ is chosen based on a criterion given later in the proof.

Consider v that is identical to v^+ on all clocks but for t_p, for which we set $\mathsf{v}(t_p) = \mathsf{v}^+(t_p) - d$. In other words $\mathsf{v}^+ = \mathsf{v} +_p d$. We find it convenient to use the notation $\mathsf{v} = \mathsf{v}^+ -_p d$. It suffices to show that $\mathsf{v} \in \mathsf{Z}$, as this will imply $\mathsf{v}^+ \in \mathsf{Z}^+$. Our choice of d ensures $\mathsf{v}(t_p - x) \preccurlyeq_{xt_p} c_{xt_p}$ for all constraints coming from edges $x \xrightarrow{\preccurlyeq_{xt_p} c_{xt_p}} t_p$ of G_Z, and $\mathsf{v}(x - y) \preccurlyeq_{yx} c_{yx}$ corresponding to edges $y \xrightarrow{\preccurlyeq_{yx} c_{yx}} x$ for all $x, y \in X' \setminus \{t_p\}$. The problem is to show $\mathsf{v}(x - t_p) \preccurlyeq_{t_p x} c_{t_p x}$ for each $x \in X' \setminus \{t_p\}$. Equivalently we want

$$\text{to show:} \quad (\leq, \mathsf{v}(t_p - x)) + (\preccurlyeq_{t_p x}, c_{t_p x}) \geq (\leq, 0) \tag{4.1}$$

Let w be the variable which corresponds to the chosen d: that is, either $d = d_1 = \mathsf{v}^+(t_p - w) - c_{wt_p}$ and $\preccurlyeq_{wt_p} = \leq$, or $d = d_2 = \mathsf{v}^+(t_p - w) - c_{wt_p} + \varepsilon$ and $\preccurlyeq_{wt_p} = <$. By our choice of v, we have:

$$\mathsf{v}(t_p - w) = \begin{cases} c_{wt_p} & \text{if } \preccurlyeq_{wt_p} \text{ is } \leq \\ c_{wt_p} - \varepsilon & \text{if } \preccurlyeq_{wt_p} \text{ is } < \end{cases} \tag{4.2}$$

Rewriting (4.1) in terms of w, we want to show:

$$(\leq, \mathsf{v}(t_p - w)) + (\leq, \mathsf{v}(w - x)) + (\prec_{t_p x}, c_{t_p x}) \geq (\leq, 0) \tag{4.3}$$

We have already seen that $\mathsf{v}(x-w) \prec_{wx} c_{wx}$. Moreover, due to the canonicity of G_{Z}, the weight $(\prec_{wx}, c_{wx})$ of the edge $w \to x$ is smaller than or equal to the weight $(\prec_{wt_p}, c_{wt_p}) + (\prec_{t_p x}, c_{t_p x})$ of the path $w \to t_p \to x$. This gives $(\leq, \mathsf{v}(x - w)) \leq (\prec_{wt_p}, c_{wt_p}) + (\prec_{t_p x}, c_{t_p x})$. Adding $(\leq, \mathsf{v}(w - x))$ to both sides of this inequality results in:

$$(\leq, 0) \leq (\leq, \mathsf{v}(w - x)) + (\prec_{wt_p}, c_{wt_p}) + (\prec_{t_p x}, c_{t_p x}) \tag{4.4}$$

Substitute the value of $\mathsf{v}(t_p - w)$ from (4.2) to the left hand side of (4.3). When $\prec_{wt_p}$ is $\leq$, this substitution gives the right hand side of (4.3) and hence we can conclude the inequality (4.3).

We are left with the case when $\prec_{wt_p}$ is $<$. In this case, the left hand side of (4.3) becomes $(\leq, c_{wt_p} - \varepsilon) + (\leq, \mathsf{v}(w - x)) + (\prec_{t_p x}, c_{t_p x})$. Here is where the choice of $\varepsilon > 0$ matters. Since $\prec_{wt_p}$ is $<$, (4.4) entails $\mathsf{v}(x-w)+c_{wt_p}+c_{t_p x} > 0$ for every $x \in \widetilde{X}' \setminus \{t_p\}$. Choose $\varepsilon := \frac{1}{2} \min\{\mathsf{v}(x - w) + c_{wt_p} + c_{t_p x} \mid x \in \widetilde{X}' \setminus \{t_p\}\}$. With this choice, we have $c_{wt_p} - \varepsilon + \mathsf{v}(w - x) + c_{t_p x} > 0$ thereby allowing us to conclude (4.3) also for this case. □

Lemma 4.4. *For each guard g, the set of local valuations satisfying g is a local zone.*

Proof. The set of constraints defining the offset zone Z_g is obtained by extending the set of constraints of Z with constraints $t_p - \widetilde{x} \sim c$, for every constraint of the form $x \sim c$ in g, where $\widetilde{x} \in \widetilde{X}_p$. □

Lemma 4.5. *For a set of clocks R, and a local zone Z, the set $[R]\mathsf{Z}$ is a local zone.*

Proof. We will show that the set of local valuations obtained on applying the reset operation to a local zone continues to be a local zone. Let G_{Z} be the canonical distance graph of Z. Note that we use the term *removing* an edge $x \to y$ from a distance graph to say that the value of this edge is assigned as $(<, \infty)$. We say a path is *lighter* than another to indicate that the weight of the former path is less than the weight of the latter path. In the same spirit, we use the term *lightest path* from x to y to refer to the path with the minimum weight from x to y.

Let G'_1 be the distance graph obtained by removing from G_{Z} all edges involving $\widetilde{x}$ and adding the edges $\widetilde{x} \xrightarrow{(\leq,0)} t_p$ and $t_p \xrightarrow{(\leq,0)} \widetilde{x}$, for each clock $x \in R$ such that $x \in X_p$. Let G_1 be the distance graph obtained by canonicalizing G'_1. We will show that the set of valuations defined by G_1 is $[R]\mathsf{Z}$, i.e., $[\![G_1]\!] = [R]\mathsf{Z}$.

Consider edges of the form $y \to z$ in G_1, where $y, z \notin R$. Notice that y and z could be reference clocks as well. We now show that the weight of such edges does not change from G_{Z} to G_1. First, observe that the first step in this transformation, i.e., removal of edges, cannot lead to lighter paths in distance graphs. Next, we inspect whether the newly added edges of weight $(\leq, 0)$ between $\widetilde{x}$ and t_p (from $\widetilde{x}$ to t_p and t_p to $\widetilde{x}$) where $x \in R$ can contribute to a lighter path from y to z in G_1'. Suppose that there was a lighter path from y to z using such a newly added edge between $\widetilde{x}$ and t_p. Consider the lightest such path from y to z.

- Suppose that the new lighter path used the $\widetilde{x} \xrightarrow{(\leq,0)} t_p$ edge. Since there are no other incoming edges to $\widetilde{x}$, the path cannot reach $\widetilde{x}$ without first going to t_p. However, this would imply that this lighter path is of the form $y \to \cdots u \to t_p \xrightarrow{(\leq,0)} \widetilde{x} \xrightarrow{(\leq,0)} t_p \to s \to \cdots z$. Removing the part $t_p \xrightarrow{(\leq,0)} \widetilde{x} \xrightarrow{(\leq,0)} t_p$ of weight 0 yields a path $y \to \cdots u \to s \to \cdots z$ in G_{Z} whose weight is the same. But this is a contradiction, as we have taken the shortest path.
- Suppose that the new lighter path is of the form $y \to \cdots t_p \to \widetilde{x} \to u \to \cdots \to z$. Recall that the only outgoing edge from $\widetilde{x}$ in G_1' is to t_p. As a consequence, if there is an edge $\widetilde{x} \to u$ in G_1, it is because of a path $\widetilde{x} \xrightarrow{(\leq,0)} t_p \to u$. But this implies that our path can be rewritten as $y \to \cdots t_p \xrightarrow{(\leq,0)} \widetilde{x} \xrightarrow{(\leq,0)} t_p \to u \to \cdots \to z$. Using the same argument as in the previous case, we arrive at a contradiction.

Thus, the weight of an edge $y \to z$ such that $y, z \notin R$ does not change from G_{Z} to G_1.

We will now show that $[\![G_1]\!] = [R]\mathsf{Z}$

$[R]\mathsf{Z} \subseteq [\![G_1]\!]$**:** Pick $\mathsf{v}' \in [R]\mathsf{Z}$. We have $\mathsf{v}'(t_p) = \mathsf{v}(t_p)$ for all reference clocks t_p, $\mathsf{v}'(t_p) - \mathsf{v}'(\widetilde{x}) = \mathsf{v}(t_p) - \mathsf{v}(\widetilde{x})$ if $x \notin R$ and $x \in X_p$ and $\mathsf{v}'(t_p) - \mathsf{v}'(\widetilde{x}) = 0$ if $x \in R$ and $x \in X_p$. Observe that v' satisfies all the constraints of G_1. Hence, $[R]\mathsf{Z} \subseteq [\![G_1]\!]$.

$[\![G_1]\!] \subseteq [R]\mathsf{Z}$**:** Consider a valuation $\mathsf{v} \in [\![G_1]\!]$. We construct a new distance graph G' whose edges are as follows:

- For clocks $x, y \notin R$, G' has the edge $x \xrightarrow{(\leq,\mathsf{v}(y-x))} y$. Note that x and y could be reference clocks here.
- For clocks $x \notin R$ such that $x \in X_p$, G' has the edges $t_p \xrightarrow{(\leq,\mathsf{v}(\widetilde{x})-\mathsf{v}(t_p))} \widetilde{x}$ and $\widetilde{x} \xrightarrow{(\leq,\mathsf{v}(t_p)-\mathsf{v}(\widetilde{x}))} t_p$.

- For clocks $x \in R$, there are no edges involving $\widetilde{x}$ in G'. This signifies that the weight of such an edge is $(\leq, \infty)$.

Notice that $\mathsf{v} \in [\![G']\!]$. Also, observe that G' is a canonical zone.

Observe that a solution to $G_{\mathsf{Z}} \cap G'$ gives a valuation v' such that $\mathsf{v}' \in \mathsf{Z}$ and v is obtained by a reset from v', and hence will imply $\mathsf{v} \in [R]\mathsf{Z}$. Now, we need to show that $G_{\mathsf{Z}} \cap G'$ is non-empty. Suppose that there is a negative cycle in $G_{\mathsf{Z}} \cap G'$. Since there were no negative cycle in G_{Z} or G', the new negative cycle should contain edges from both G_{Z} and G'. Further, since both G_{Z} and G' are canonical, the negative cycle should alternate between edges of G_{Z} and G'.

Consider an edge $x \to y$, where $x, y \notin R$. Recall that the weight of such an edge does not change from G_{Z} to G_1. Since $\mathsf{v} \in [\![G_1]\!]$, we know that

$$\mathsf{v}(y) - \mathsf{v}(x) \leq c, \tag{4.5}$$

where $(\leq, c)$ is the weight of the edge $x \to y$ in G_{Z}. Further, observe that the weight of the edge $x \to y$ in G' has weight $\mathsf{v}(y) - \mathsf{v}(x)$. From 4.5, we can see that this edge in $G_{\mathsf{Z}} \cap G'$ has to be from G'. Thus, all edges in $G_{\mathsf{Z}} \cap G'$ of the form $x \to y$ where $x, y \notin R$ come from G'. This implies that the negative cycle cannot be limited to the clocks that were not reset.

Also, observe that since there are no edges in G' involving $\widetilde{x}$ where $x \in R$, any edges involving these clocks must come from G_{Z}. Suppose our negative cycle contained a clock $\widetilde{x}$ such that $x \in R$. By our criterion for the negative cycle, either the incoming or the outgoing edge associated to $\widetilde{x}$ should come from G', which we know is not possible as there are no edges associated to s in G'. This implies that the negative cycle cannot involve clocks that are reset.

Let the negative cycle be of the form

$$x \to \cdots \to y \to t_p \to z \to u \to \cdots \to x$$

where t_p is a reference clock and all other clocks could be either reference clocks or offset clocks that were not reset.

Suppose that the edge $t_p \to z$ comes from G'. Then, the edge $z \to u$ should be from G_{Z}. However, since $u \notin R$, we know that the edge $z \to u$ in $G_{\mathsf{Z}} \cap G'$ comes from G', which is contrary to our requirement. Thus, the edge $t_p \to z$ comes from G_{Z}. As a consequence, the incoming edge to t_p, namely $y \to t_p$, should be from G'.

We have already seen that all the edges of the form $x \to y$ such that $x, y \notin R$ can only be from G'. Since G' is canonical we can replace all these edges by a single edge to get the negative cycle

$$x \to t_p \to z \to x$$

Further, since $x \to t_p$ and $z \to x$ are both from G', we can replace it by a single edge. Thus, the situation boils down to a negative cycle of just 2 edges of the form

$$z \xrightarrow{(\preccurlyeq,k_1)} t_p \xrightarrow{(\preccurlyeq,k_2)} z$$

where the edge $z \xrightarrow{(\preccurlyeq_1,k_1)} t_p$ is from G_{Z} and the edge $t_p \xrightarrow{(\preccurlyeq_2,k_2)} z$ is from G'. From our assumption, since this is a negative cycle, we have $(\preccurlyeq_1, k_1) + (\preccurlyeq_2, k_2) < (\leq, 0)$. Since we know that $k_2 = \mathsf{v}(z) - \mathsf{v}(t_p)$, this implies

$$\mathsf{v}(z) - \mathsf{v}(t_p) + k_1 < 0$$

We know that the edges between z and t_p in G_1 is unchanged from G_{Z}, since $z \notin R$. Further, since we know that $\mathsf{v} \in [\![G_1]\!]$, we have $\mathsf{v}(t_p) - \mathsf{v}(z) \leq k_1$, which implies

$$\mathsf{v}(z) - \mathsf{v}(t_p) + k_1 \geq 0$$

This is a contradiction. Thus, our assumption that there was a negative cycle was wrong. □

The operations of local time elapse, guard intersection, and reset, enable us to describe a local step $(q, \mathsf{Z}) \xrightarrow{b} (q', \mathsf{Z}')$ on the level of local zones. This is done in the same way as for global zones. Observe that a local step is indexed only by an action, as the aspect of time is taken care of by the local time elapse operation.

Definition 4.2. There is a transition $(q, \mathsf{Z}) \xrightarrow{b} (q', \mathsf{Z}')$ provided for each process $p \in \mathsf{dom}(b)$ its unique transition on b is $(q(p), g_p, R_p, q'(p))$ for some g_p and R_p, $\mathsf{Z}' = \text{local-elapse}(\mathsf{Z}_2)$ and $\mathsf{Z}' \neq \emptyset$ with $\mathsf{Z}_2 = [\bigcup_{p \in \mathsf{dom}(b)} R_p]\mathsf{Z}_1$ and $\mathsf{Z}_1 = \mathsf{Z} \cap \bigcap_{p \in \mathsf{dom}(b)} \mathsf{Z}_{g_p} \cap \{t_{p_1} = t_{p_2} \mid p_1, p_2 \in \mathsf{dom}(b)\}$.

For a sequence of actions u we write $(q, \mathsf{Z}) \xrightarrow{u} (q', \mathsf{Z}')$ for a sequence of transitions indexed by elements of u.

4.3. Local zone graph

Equipped with the notion of local zones, we can now construct a local zone graph and show that it is sound and complete for reachability testing. We say a local zone is *time-elapsed* if $\mathsf{Z} = \text{local-elapse}(\mathsf{Z})$.

Definition 4.3 (Local zone graph)**.** For a network of timed automata $\mathcal{N}$ the *local zone graph* of $\mathcal{N}$, denoted $\mathsf{LZG}(\mathcal{N})$, is a transition system whose nodes are of the form (q, Z) where q is a state of the network and Z is a time elapsed local zone, and whose transitions are steps $(q, \mathsf{Z}) \xrightarrow{b} (q', \mathsf{Z}')$. The initial node (q_0, Z_0) consists of the initial state q_0 of the network and the local zone $\mathsf{Z}_0 = \text{local-elapse}(\{\mathsf{v}_0\})$, v_0 is the initial (local) valuation.

Next, we prove the pre/post properties of runs on local zones.

Lemma 4.6 (Pre and post properties of runs on local zones)**.** *Let u be a sequence of actions.*

- *If $(q, \mathsf{v}) \xrightarrow{u} (q', \mathsf{v}')$ and $\mathsf{v} \in \mathsf{Z}$ for some time-elapsed local zone Z, then $(q, \mathsf{Z}) \xrightarrow{u} (q', \mathsf{Z}')$ and $\mathsf{v}' \in \mathsf{Z}'$ for some local zone Z'.*
- *If $(q, \mathsf{Z}) \xrightarrow{u} (q', \mathsf{Z}')$ and $\mathsf{v}' \in \mathsf{Z}'$ then $(q, \mathsf{v}) \xrightarrow{u} (q', \mathsf{v}')$, for some $\mathsf{v} \in \mathsf{Z}$.*

Proof. Proof follows by induction on the length of u.

For the pre property: Suppose that u is a single action a. Then, $(q, \mathsf{v}) \xrightarrow{a} (q', \mathsf{v}')$ implies there is a sequence of local steps: $(q, \mathsf{v}) \xrightarrow{\Delta} (q, \mathsf{v}_1) \xrightarrow{a} (q', \mathsf{v}_2) \xrightarrow{\Delta'} (q', \mathsf{v}')$. Since $\mathsf{v} \in \mathsf{Z}$ and Z is local time elapsed, we have $\mathsf{v}_1 \in \mathsf{Z}$. By definition of $(q, \mathsf{Z}) \xrightarrow{a} (q', \mathsf{Z}')$ we get $\mathsf{v}' \in \mathsf{Z}'$. The induction step follows by a similar argument, and noting the fact that Z' is local time-elapsed in $(q, \mathsf{Z}) \xrightarrow{a} (q', \mathsf{Z}')$.

For the post property: When u is a single action, the definition entails that there is a $\mathsf{v} \in \mathsf{Z}$ such that $(q, \mathsf{v}) \xrightarrow{a}\xrightarrow{\Delta} (q', \mathsf{v}')$. When $u = a_1 \dots a_n$, consider the sequence of zones $(q, \mathsf{Z}) \xrightarrow{a_1} (q_1, \mathsf{Z}_1) \cdots (q_{n-1}, \mathsf{Z}_{n-1}) \xrightarrow{a_n} (q', \mathsf{Z}')$, use a similar argument to obtain a $\mathsf{v}_{n-1} \in \mathsf{Z}_{n-1}$ and then the induction hypothesis for the shorter sequence $a_1 \dots a_{n-1}$. □

Directly from Lemma 4.6 we obtain the main property of local zone graphs which allows us to use it for reachability testing.

Theorem 4.1. *For a given network of timed automata $\mathcal{N}$, there is a run of the network reaching a state q iff for some non-empty local zone Z, node (q, Z) is reachable in $\mathsf{LZG}(\mathcal{N})$ from its initial node.*

Remark. *The proof of Theorem 4.1 is based on Lemma 3.9 which says that if there is a local run to a configuration (q, v) where v is a synchronised valuation, then the state q is reachable. The "non-empty" in the statement of Theorem 4.1 is meant to highlight the requirement that the zone contains at least one synchronized local valuation. This is because if the zone contains at least one local valuation, then it also contains a synchronized valuation (obtained by synchronizing this local valuation).*

Notice that $\mathsf{LZG}(\mathcal{N})$ may still be infinite and it cannot be used directly for reachability checking. This problem will be addressed in Chapter 5.

4.4. Commutativity in the local zone graph

In this section, we discuss some important properties of the local zone graph with respect to concurrency. In Section 3.4, we showed that in local

time semantics for networks of timed automata, two actions with disjoint domains are independent. Specifically, we showed that two actions with disjoint domains satisfied the diamond property as well as the forward diamond property (see Definition 2.48.) In this section, we investigate if these properties extend to the setting of local zone graphs.

Diamond property: First, we examine if the diamond property holds for actions with disjoint domains in the local zone graph.

Lemma 4.7. *Let* $\mathsf{dom}(a) \cap \mathsf{dom}(b) = \emptyset$. *If* $(q, \mathsf{Z}) \xrightarrow{ab} (q', \mathsf{Z}')$, *then* $(q, \mathsf{Z}) \xrightarrow{ba} (q', \mathsf{Z}')$.

Proof. Suppose that $(q, \mathsf{Z}) \xrightarrow{ab} (q', \mathsf{Z}'_{ab})$. We will show that there is $(q, \mathsf{Z}) \xrightarrow{ba} (q', \mathsf{Z}'_{ba})$, and that $\mathsf{Z}'_{ab} \subseteq \mathsf{Z}'_{ba}$. By symmetry this will show the lemma.

Take $\mathsf{v}' \in \mathsf{Z}'_{ab}$. Using the backward (post) property of steps on zones (Lemma 4.6) we get a run:

$$(q, \mathsf{v}) \xrightarrow{a} (q_a, \mathsf{v}_a) \xrightarrow{\Delta_a} (q_a, \mathsf{v}'_a) \xrightarrow{b} (q', \mathsf{v}_b) \xrightarrow{\Delta_b} (q', \mathsf{v}'),$$

where $\mathsf{v} \in \mathsf{Z}$. Using commutation on the level of runs, Lemma 3.4, we get a run

$$(q, \mathsf{v}) \xrightarrow{\Delta'_b} (q, \mathsf{v}'_b) \xrightarrow{b} (q_b, \mathsf{v}''_b) \xrightarrow{\Delta'_a} (q_b, \mathsf{v}''_a) \xrightarrow{a} (q', \mathsf{v}'''_a) \xrightarrow{\Delta''_a} (q', \mathsf{v}')$$

Now using the forward (pre) property of steps on zones from Lemma 4.6 we obtain that Z_{ba} exists and $\mathsf{v}' \in \mathsf{Z}_{ba}$. □

Thus, it is clear that the diamond property holds for actions with disjoint domains in the local zone graph. Extending Lemma 4.7 to longer sequences of actions, we get the following property, which states that if a run σ is feasible from a zone in the local zone graph, then all runs equivalent to σ (in the sense of $\sim$ equivalence as given in Definition 3.1) are feasible from that zone and lead to the same zone.

Corollary 4.1. *If* $(q, \mathsf{Z}) \xrightarrow{u} (q', \mathsf{Z}')$ *and* $u \sim w$, *then* $(q, \mathsf{Z}) \xrightarrow{w} (q', \mathsf{Z}')$.

Forward diamond property: We show that actions with disjoint domains do not satisfy the forward diamond property in the local zone graph. In Lemma 4.8, we point out that even if two actions a and b have disjoint domains and are individually enabled from a local zone (q, Z), the sequence ab need not be enabled from (q, Z). Observe that this is in contrast to the situation at the valuation level for local time semantics as given in Lemma 3.4.

Lemma 4.8. *There exists a local zone* (q, Z) *and actions* a *and* b *such that* $\mathsf{dom}(a) \cap \mathsf{dom}(b) = \emptyset$, *both actions* a *and* b *are feasible from* (q, Z) *but neither* ab *nor* ba *is feasible from* (q, Z).

Proof. Let P_1 and P_2 be processes with clocks $\{x_1, y_1\}$ and $\{x_2, y_2\}$ respectively. Consider a transition sequence consisting of four synchronized actions:

$$\rho: \xrightarrow[\{y_2\}]{} \xrightarrow[\{x_2\}]{} \xrightarrow[\{y_1\}]{x_2 \geq 1} \xrightarrow[\{x_1\}]{y_2 \leq 3}$$

Since all actions are synchronized, the local zone obtained after the sequence ρ is Z_ρ which will satisfy the following constraints (recall that the clocks maintain the timestamps of the last resets):

$$0 \leq \widetilde{y}_2 \leq \widetilde{x}_2 \leq \widetilde{y}_1 \leq \widetilde{x}_1 \quad \text{and} \quad \widetilde{x}_1 - \widetilde{y}_2 \leq 3 \quad \text{and} \quad \widetilde{y}_1 - \widetilde{x}_2 \geq 1$$

Because of these interleaved constraints, for each valuation $\mathsf{v} \in \mathsf{Z}_\rho$:

$$\begin{aligned} \mathsf{v}(\widetilde{x}_1) - \mathsf{v}(\widetilde{y}_1) = 2 &\implies \mathsf{v}(\widetilde{x}_2) - \mathsf{v}(\widetilde{y}_2) = 0 \\ \text{and } \mathsf{v}(\widetilde{x}_2) - \mathsf{v}(\widetilde{y}_2) = 2 &\implies \mathsf{v}(\widetilde{x}_1) - \mathsf{v}(\widetilde{y}_1) = 0 \end{aligned}$$

Now, consider two local actions with guards: $g_1 : y_1 \geq 3 \wedge x_1 \leq 1$ and $g_2 : y_2 \geq 3 \wedge x_2 \leq 1$. Guard g_1 implies that $\widetilde{x}_1 - \widetilde{y}_1 \geq 2$ and similarly g_2 implies that $\widetilde{x}_2 - \widetilde{y}_2 \geq 2$. Intersecting Z_ρ with g_1 gives valuations in which $\mathsf{v}(\widetilde{x}_1) - \mathsf{v}(\widetilde{y}_1) \geq 2$ and $\mathsf{v}(\widetilde{x}_2) - \mathsf{v}(\widetilde{y}_2) \leq 0$. Similarly, intersecting Z_ρ with g_2 gives the other symmetric combination. Hence from the set obtained by taking action g_1, action g_2 is not enabled and vice versa. However, both g_1 and g_2 are enabled at Z_ρ. □

Thus, combining Lemma 4.8 and Corollary 4.1, we can conclude the following: Disjoint domains is not a sufficient condition to conclude that two actions are independent. In other words, just because two actions a and b have disjoint domains, we cannot postpone the execution of one of them in favor of the other. However, if we know that ab is feasible, then ba is feasible as well.

4.5. Aggregate zones in local zone graph

We know from Corollary 4.1 that starting from a local zone all equivalent interleavings of a sequence of actions u end up in the same local zone. This is in stark contrast to the global zone graph, where each interleaving results in a possibly different global zone (see Figure 4.1). Let

$$\mathsf{MZ}(q, Z, u) = \{v' \mid \exists v \in Z,\ \exists w,\ w \sim u \ \text{ and } (q, v) \overset{w}{\Longrightarrow} (q', v')\}$$

denote the union of all these global zones.

Salah et al. [SBM06] have shown that, surprisingly, $\mathsf{MZ}(q, Z, u)$ is always a global zone. We call it *aggregated zone*, and the notation MZ is in the memory of Oded Maler. In the same work, this observation was extended to

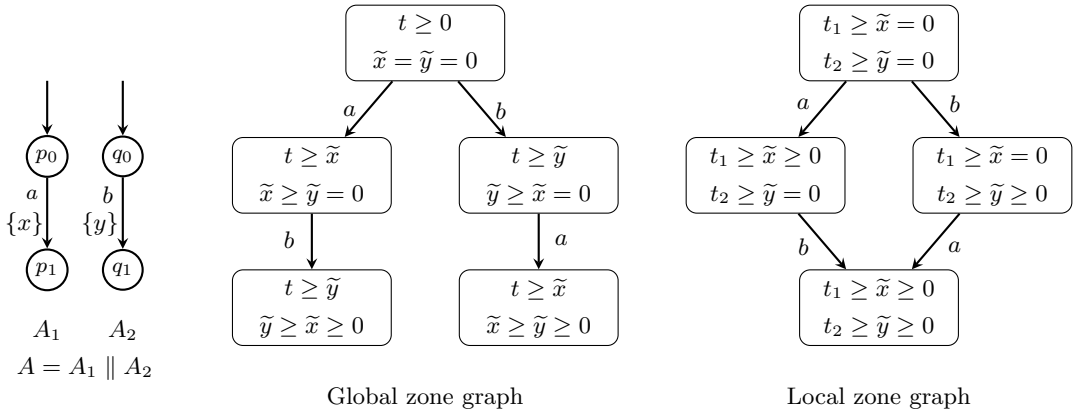


Figure 4.1: Local zone graph and global zone graph of a network. If two actions of a given sequence σ reset clocks, then two interleavings of σ with a different ordering of these actions lead to different nodes of the global zone graph (if both of them are feasible.).

an algorithm for *acyclic timed automata* that from time to time merged zones reached by equivalent paths to a single global zone. We prove below that this aggregated zone can, in fact, be obtained directly in the local zone graph: the aggregated (global) zone is exactly the set of synchronized valuations obtained after executing u in the local zone semantics. Here we need some notation: let Z be a global zone and Z a local zone; define

$$\mathsf{sync}(\mathsf{Z}) = \{\mathsf{v} \in \mathsf{Z} \mid \mathsf{v} \text{ is synchronized}\}$$
$$\mathsf{local}(Z) = \{\mathsf{local}(v) \mid v \in Z\}$$
$$\mathsf{global}(\mathsf{sync}(\mathsf{Z})) = \{\mathsf{global}(\mathsf{v}) \mid \mathsf{v} \in \mathsf{sync}(\mathsf{Z})\}$$

Lemma 4.9. *For each global zone Z and local zone Z:* $\mathsf{sync}(\mathsf{Z})$ *and* $\mathsf{local}(Z)$ *are local zones and* $\mathsf{global}(\mathsf{sync}(\mathsf{Z}))$ *is a global zone.*

Proof. $\mathsf{sync}(\mathsf{Z})$ is the local zone $\mathsf{Z} \wedge \bigwedge_{i,j}(t_i = t_j)$; $\mathsf{local}(Z)$ is the local zone obtained by replacing t with some t_i in each constraint, and adding the constraints $\bigwedge_{i,j}(t_i = t_j)$; $\mathsf{global}(\mathsf{sync}(\mathsf{Z}))$ is obtained by replacing each t_i with t in each constraint of $\mathsf{sync}(\mathsf{Z})$. □

Theorem 4.2. *Consider a state q, a sequence of actions u and a time elapsed global zone Z. Consider the local zone* $\mathsf{Z} = \text{local-elapse}(\mathsf{local}(Z))$. *If* $(q, \mathsf{Z}) \xrightarrow{u} (q', \mathsf{Z}')$, *we have* $\mathsf{MZ}(q, Z, u) = \mathsf{global}(\mathsf{sync}(\mathsf{Z}'))$, *otherwise* $\mathsf{MZ}(q, Z, u) = \emptyset$.

Proof. Pick $v' \in \mathsf{MZ}(q, Z, u)$. There exists $w \sim u$, $v \in Z$ and a global run $(q, v) \xRightarrow{w} (q', v')$. From Lemma 3.9, there exists a local run $(q, \mathsf{local}(v)) \xrightarrow{w}$

$(q', \mathsf{local}(v'))$. By assumption, $\mathsf{local}(v) \in \mathsf{Z}$. Hence from the pre property of local zones (Lemma 4.6), there exists $(q, \mathsf{Z}) \xrightarrow{w} (q', \mathsf{Z}_w)$ such that $\mathsf{local}(v') \in \mathsf{Z}_w$. As $\mathsf{local}(v')$ is synchronized, we get $\mathsf{local}(v') \in \mathsf{sync}(\mathsf{Z}_w)$. But, by Corollary 4.1, $\mathsf{Z}_w = \mathsf{Z}'$. This proves $\mathsf{local}(v') \in \mathsf{sync}(\mathsf{Z}')$ and hence $v' \in \mathsf{global}(\mathsf{sync}(\mathsf{Z}'))$.

For the other direction take $v' \in \mathsf{global}(\mathsf{sync}(\mathsf{Z}'))$. As $(q, \mathsf{Z}) \xrightarrow{u} (q', \mathsf{Z}')$, by post property of local zones (Lemma 4.6) there is a local run $(q, \mathsf{v}_u) \xrightarrow{u} (q', \mathsf{local}(v'))$ for some $\mathsf{v}_u \in \mathsf{Z}$. Since $\mathsf{v}_u \in \mathsf{Z}$, it is obtained by a local time elapse from some $\mathsf{v} \in \mathsf{local}(Z)$. Hence v is synchronized and $\mathsf{global}(\mathsf{v}) \in Z$. From Lemma 3.9 we get that for some $w \sim u$ there is a global run $(q, \mathsf{global}(\mathsf{v})) \xRightarrow{w} (q', v')$. Hence $v' \in \mathsf{MZ}(q, Z, u)$. □

Theorem 4.2 gives an efficient way to compute aggregated zones: it is sufficient to compute local zone graphs. Computing local zone graphs is not more difficult than computing global zone graphs. But, surprisingly, on the level of zones, the combinatorial explosion due to interleaving does not occur in local zone graphs, thanks to the theorem above. Hence, this gives an incentive to work with local zone graphs instead of global zone graphs.

This contrasts with the aggregation algorithm in [SBM06] which requires to store all the paths to a global zone and detect situations where zones can be merged, that is, when all the equivalent permutations have been visited. Another important limitation of the algorithm from [SBM06] is that it can only be applied to acyclic zone graphs. If local zone graphs can be computed for general timed automata (which contain cycles), we can get to use the aggregation feature for all networks (and not only acyclic ones). To do this, there is still a major problem left: local zone graphs could be infinite when the automata contain cycles.

Chapter 5

Making local zone graphs finite

In Chapter 4, we introduced local zone graphs for networks of timed automata. We proved in Theorem 4.1 that local zone graphs are sound and complete w.r.t. reachability, i.e., there is a path from the initial node to a node (q, Z) in the local zone graph of a timed automaton A if and only if q is reachable in A. We also showed that the local zone graph of a network of timed automata has the following useful property: starting from a node of the local zone graph, all equivalent interleavings of a sequence of actions end up in the same node. We pointed out that this is in stark contrast to the case in the global zone graph, where each interleaving results in a potentially different global zone. We showed that this property makes it an attractive option to use the exploration of local zone graph, rather than the global zone graph, to answer the reachability problem for networks of timed automata. Unfortunately, local zone graphs may not be finite and an algorithm that proceeds by exploring the local zone graph is not guaranteed to terminate. To ensure termination of such an algorithm, we need some way to compute a finite truncation of local zone graph, whose exploration is sufficient.

We face a similar problem in the case of global zone graphs, which are also infinite in general. The standard approach to make a global zone graph finite involves using a node covering relation between the nodes of the global zone graph (discussed in detail in Section 2.8). In order to use the local zone graph to answer the reachability problem for networks of timed automata, we need such a covering relation between nodes of the local zone graph.

In this chapter we first examine some of the solutions proposed in the literature to obtain a node covering relation between nodes of the local zone graph. We point out that these solutions either have some fatal flaws or are not effectively computable. We then go on to propose a new node covering relation for nodes of the local zone graph that is based on a relation between local zones that we call *sync-subsumption*. It allows us to define a

transition system called *local sync graph* that is the local zone graph reduced by applying this new node covering relation. We propose an algorithm that uses the exploration of local sync graphs to answer the reachability problem for networks of timed automata.

Applying a partial order reduction procedure to the exploration of a local sync graph is one way to make this procedure even more efficient. However, we do not know how to compute the independence relation of actions of local sync graphs, and as a consequence, do not know how to apply partial order reduction to local sync graphs.

The results in this chapter appeared in [GHSW19].

5.1. Approaches to get finiteness for local zone graphs

The standard approach to make a global zone graph finite involves using a *subsumption relation* between global zones. The subsumption relation is defined using an abstraction operator $\mathfrak{a}$ (see Definition 2.35). The relation $\sqsubseteq^{\mathfrak{a}}_{LU}$ (discussed in detail in Section 2.8) is one such subsumption relation between global zones that uses the $\mathfrak{a}_{\preccurlyeq LU}$ abstraction operator. Abstraction operators are usually based on a simulation relation between global valuations (see Definition 2.18). Abstractly, a simulation relation is a relation between valuations that clubs together those valuations that are indistinguishable with respect to runs from them. Recall that simulation relations between global valuations are usually parameterized by certain maximum constants occurring in guards, for instance the region equivalence for global valuations (see Definition 2.16) which is parameterized by the maximum constant M occurring in guards, and the LU preorder (see Definition 2.39) which is parameterized by L and U, the maximum constants respectively used in a lower bound and upper bound guards. Additionally, if it is known that there are only finitely many distinct sets $\mathfrak{a}(\mathcal{Z})$ where $\mathcal{Z}$ is a global zone, the subsumption relation based on $\mathfrak{a}$ can be used to obtain an effective procedure to answer the reachability problem for timed automata, as was shown in Section 2.8.

Observe that the existence of a simulation relation for global valuations (LU preorder) was crucial to obtain an abstraction operator, $\mathfrak{a}_{\preccurlyeq LU}$, for global zones (Definition 2.40). The fact that $\mathfrak{a}_{\preccurlyeq LU}$ over global zones is a simulation relation of finite index was used to obtain conditions to guarantee termination of exploration of the global zone graph.

In this section, we present some of the solutions in standard literature to the problem of obtaining a subsumption relation for local zones. For each of these solutions, we show that there are technical problems that restrict its effectiveness.

5.1.1 Catch-up equivalence

In the work that introduced local time semantics and local zone graphs, Bengtsson et al. [BJLY98] define an equivalence between local valuations, referred to as *catch-up equivalence*. They also propose an extension of catch-up equivalence to local zones. In their work, they show that catch-up equivalence for local zones is a simulation relation of finite index. Then, following the same ideas as for global zones, one can hope to construct a finite local zone graph using a subsumption relation between local zones based on catch-up equivalence. Since this graph is guaranteed to be finite, this could be used to answer the reachability problem. In this section we present the definition of catch-up equivalence and discuss the problems associated with this approach.

We say two synchronized local valuations v and v' are region-equivalent if $\overline{v} \equiv_M \overline{v}'$, where $\overline{v} = \mathsf{global}(\mathsf{v})$ and $\overline{v}' = \mathsf{global}(\mathsf{v}')$ (for region equivalence, see Definition 2.16). Equivalence classes of synchronized local valuations are called *synchronized regions*. Bengtsson et al. [BJLY98] propose to extrapolate the region equivalence that is defined on synchronized valuations to non-synchronized valuations by introducing the notion of *catch-up transitions*.

Recall that in local time semantics, each process has a different reference clock. Intuitively, a catch-up transition is one that allows processes which are lagging behind w.r.t. to the reference clock value, to catchup with the processes having higher values of reference clocks. Essentially, by doing this, we are allowing the system to move to a state that is "more synchronized in time".

Definition 5.1 (Catch-up transition)**.** A local delay transition $(q, \mathsf{v}) \xrightarrow{\delta}_i (q, \mathsf{v}')$ is called a *catch-up transition* if the maximum value of reference clocks at v' is not greater than the maximum value of reference clocks at v, i.e.,

$$\max_{p \in \mathsf{Proc}} \mathsf{v}'(t_p) \leq \max_{p \in \mathsf{Proc}} \mathsf{v}(t_p)$$

For example, consider a network $\mathcal{N}_1$ of two timed automata and a local valuation v of $\mathcal{N}_1$ where: $\mathsf{v}(t_1) = 10, \mathsf{v}(t_2) = 12$. Now consider the valuation v_1 obtained by executing a local delay of $\delta = 2$ in process A_1. We have $\mathsf{v}_1(t_1) = \mathsf{v}_1(t_2) = 12$. Here, δ is an example of a catch-up transition. However, observe that not all local delay transitions are catch-up transitions. For instance, a local delay transition of $\delta' = 5$ in process A_1 from v takes us to a valuation v_2 where $\mathsf{v}_2(t_1) = 15, \mathsf{v}_2(t_2) = 12$ - here, the maximum value of reference clocks at v_2 is greater than the maximum value of reference clocks at v, and we end up with a valuation which is no more synchronized than the valuation v that we started out with. Hence, δ' is not a catch-up transition.

We denote by $R(q, \mathsf{v})$ the set of all synchronized regions that can be reached from the (state, local valuation) pair (q, v) by action transitions or catch-up transitions.

Definition 5.2 (Catch-up equivalence)**.** (q, v) and (q', v') are said to be catch-up equivalent if the set of synchronized regions reachable from both these states are the same; i.e.,

$$(q, \mathsf{v}) \sim_{catch-up} (q', \mathsf{v}') \text{ if } R(q, \mathsf{v}) = R(q', \mathsf{v}')$$

From Lemma 2.10, we know that the number of synchronized regions is finite. As a consequence, we can conclude that catch-up equivalence also has finite index.

Unfortunately, the question of effective algorithms to check for catch-up equivalence was left open in [BJLY98]. To the best of our knowledge, there is no efficient procedure to check if two local valuations are catch-up equivalent. Therefore, catch-up equivalence, though useful theoretically, cannot be used to obtain terminating procedures to check reachability via the exploration of local zone graphs.

5.1.2 Minea's approach

Building on [BJLY98], another finite abstraction of the local zone graph was proposed by Minea [Min99a, Min99b]. In this section, we discuss this approach and point out that it carries a bug, and it is not evident how to repair this bug.

The approach in [Min99a] is founded on an equivalence between local valuations along the lines of the region equivalence proposed by Alur et al. [AD94] (see Section 2.3). We present the equivalence as presented in [Min99a] below. Note that in this section, the set of all offset variables including the reference clocks is denoted as T^+.

Definition 5.3 (Equivalence defined in Section 3.7 of [Min99a])**.** Fix a network of timed automata $\mathcal{N} = \langle A_1, \ldots, A_k \rangle$. Let $c_{\max}$ be the maximum constant used in the guards of $\mathcal{N}$. Two local valuations v and v' are said to be equivalent, written as $\mathsf{v} \simeq_{\text{reg}} \mathsf{v}'$ if for all variables $\widetilde{x}, \widetilde{y} \in T^+$ (including the reference clocks):

- either $\lfloor \mathsf{v}(\widetilde{x}) - \mathsf{v}(\widetilde{y}) \rfloor = \lfloor \mathsf{v}'(\widetilde{x}) - \mathsf{v}'(\widetilde{y}) \rfloor$,
- or $\lfloor \mathsf{v}(\widetilde{x}) - \mathsf{v}(\widetilde{y}) \rfloor > c_{\max}$ and $\lfloor \mathsf{v}'(\widetilde{x}) - \mathsf{v}'(\widetilde{y}) \rfloor > c_{\max}$,
- or $\lfloor \mathsf{v}(\widetilde{x}) - \mathsf{v}(\widetilde{y}) \rfloor < -c_{\max}$ and $\lfloor \mathsf{v}'(\widetilde{x}) - \mathsf{v}'(\widetilde{y}) \rfloor < -c_{\max}$.

The equivalence is extended to configurations: $(q, \mathsf{v}) \simeq_{\text{reg}} (q', \mathsf{v}')$ if $q = q'$ and $\mathsf{v} \simeq_{\text{reg}} \mathsf{v}'$.

We now state an observation about the equivalence from Definition 5.3 made in [Min99a].

Let $\mathsf{v} \simeq_{\text{reg}} \mathsf{v}'$ w.r.t. $c_{\max}$. Then:

1. If g is a guard with a constant smaller than $c_{\max}$, then $\mathsf{v} \models g$ iff $\mathsf{v}' \models g$.
2. For each clock set R, $[R]\mathsf{v} \simeq_{\text{reg}} [R]\mathsf{v}'$.
3. For each $i \in \{1, \ldots, k\}$ and $\delta \geq 0$ there exists $\delta' \geq 0$ such that $\mathsf{v} +_i \delta \simeq_{\text{reg}} \mathsf{v}' +_i \delta'$.

Based on this observation, it is claimed that $\simeq_{\text{reg}}$ is a simulation relation. We state the proposition given in [Min99a] below.

Let $(q, \mathsf{v}) \simeq_{\text{reg}} (q, \mathsf{v}')$ be equivalent configurations.

1. If $(q, \mathsf{v}) \xrightarrow{a} (q_1, \mathsf{v}_1)$, there exists (q_1, v'_1) such that $(q, \mathsf{v}') \xrightarrow{a} (q_1, \mathsf{v}_1)$ and $(q_1, \mathsf{v}_1) \simeq_{\text{reg}} (q_1, \mathsf{v}'_1)$.
2. If $(q, \mathsf{v}) \xrightarrow{\delta}_i (q, \mathsf{v}_1)$ there exists $\delta' \in \mathbb{R}_{\geq 0}$ such that $(q, \mathsf{v}') \xrightarrow{\delta'}_i (q, \mathsf{v}'_1)$ and $(q, \mathsf{v}_1) \simeq_{\text{reg}} (q, \mathsf{v}'_1)$.

We will now show that this claim is not correct. In particular, we show that it is possible to construct a local delay transition that can distinguish two local configurations that are $\simeq_{\text{reg}}$ equivalent.

Proposition 5.1. *There exist local valuations* v *and* v' *and a delay* δ *such that* $\mathsf{v} \simeq_{\text{reg}} \mathsf{v}'$*, but for no delay* δ'*, we have* $\mathsf{v} + \delta \simeq_{\text{reg}} \mathsf{v}' + \delta'$*.*

Proof. Consider a network $\mathcal{N}_1$ of two timed automata A_1 and A_2, such that $X_1 = \{x\}$, $X_2 = \{y\}$. This gives $\widetilde{X} = \{\widetilde{x}, t_1, \widetilde{y}, t_2\}$. Let $c_{\max} = 3$. Define valuations v and v' as follows:

$$\mathsf{v} : \widetilde{x} = 0, t_1 = 0, \widetilde{y} = 0, t_2 = 4 \qquad \mathsf{v}' : \widetilde{x} = 0, t_1 = 0, \widetilde{y} = 0, t_2 = 5$$

Note that the differences between variables in v are either $0, 4$ or -4 and the corresponding differences in v' are $0, 5$ or -5. Hence by definition, $\mathsf{v} \simeq_{\text{reg}} \mathsf{v}'$. Consider valuation $\mathsf{v} +_1 2$ obtained by local delay of 2 units in process A_1 from v:

$$\mathsf{v} +_1 2 : \widetilde{x} = 0, t_1 = 2, \widetilde{y} = 0, t_2 = 4$$

Observe that in $\mathsf{v} +_1 2$, the difference $\widetilde{x} - t_1 = -2$ and $t_1 - t_2 = -2$ both of which are greater than $-c_{\max}$. We claim there is no local delay δ' such that $\mathsf{v} +_1 2 \simeq_{\text{reg}} \mathsf{v}' +_1 \delta'$. Valuation $\mathsf{v}' +_1 \delta'$ is given by $\widetilde{x} = 0, t_1 = \delta', \widetilde{y} = 0, t_2 = 5$. If $\mathsf{v} +_1 2 \simeq_{\text{reg}} \mathsf{v}' +_1 \delta'$, on the one hand, we need $\lfloor -\delta' \rfloor = -2$, hence $\delta' \leq 2$, and on the other hand, we need $\lfloor \delta' - 5 \rfloor = -2$, which entails $\delta' \geq 3$. This is not possible. □

Note that Proposition 5.1 contradicts the statement 3 of the claim made in [Min99a]. The main problem is that even from a valuation where the difference between a pair of reference clocks is bigger than the maximum constant, by executing local delays, this difference can be brought within

the maximum constant. This situation is fundamentally different from the standard semantics, where if a clock is above the maximum constant in a valuation, further delays keep it above this constant. This allowed the region equivalence to work in the global semantics.

Using the propositions given in their work, the following theorem is proposed in [Min99a].

Definition 5.4 (Maximization operator, Section 3.7 of Minea)**.** Given a canonical representation of a local zone Z, the maximized zone $\max(\mathsf{Z})$ with respect to $c_{\max}$ is obtained by the following modifications on Z. For each $\widetilde{x}, \widetilde{y} \in \widetilde{X}$ and each constraint $\phi := \widetilde{x} - \widetilde{y} \lessdot c$: if $c > c_{\max}$ remove the constraint ϕ, and if $c < -c_{\max}$, change ϕ to $\widetilde{x} - \widetilde{y} < -c_{\max}$. Canonicalize the resulting set of constraints.

Clearly the number of maximized zones is finite. Given a network $\mathcal{N}$, we can now define a transition system $\mathsf{MaxOZG}(\mathcal{N})$ as follows: nodes are of the form (q, Z) where Z is a maximized zone; there is a transition $(q, \mathsf{Z}) \xrightarrow{a} (q', \max(\mathsf{Z}'))$ if there is a transition $(q, \mathsf{Z}) \xrightarrow{a} (q', \mathsf{Z}')$ in $\mathsf{OZG}(\mathcal{N})$; the initial node is of the form $(q_0, \max(\mathsf{Z}_0))$ where (q_0, Z_0) is the initial node of $\mathsf{OZG}(\mathcal{N})$. Theorem 5 of [Min99a] (which talks about LTL model checking) can be rephrased for the question of reachability as follows. There is a run of the network $\mathcal{N}$ reaching a state q iff for some non-empty (maximized) zone Z, (q, Z) is reachable in $\mathsf{MaxOZG}(\mathcal{N})$ from its initial node. A natural implication of this theorem is the soundness and completeness of $\mathsf{MaxOZG}(\mathcal{N})$ w.r.t. reachability.

Using the observation that we made in Proposition 5.1, we are now able to state the following theorem.

Theorem 5.1. $\mathsf{MaxOZG}(\mathcal{N})$ *is not sound for reachability.*

Proof. We will make use of valuations v and v' given in the proof of Proposition 5.1. We will come up with an automaton network $\langle A_1, A_2 \rangle$ such that: (1) accepting state is not reachable; (2) valuation v' is reachable in the network and hence is present in a reachable zone; (3) maximization of the zone containing v' adds v to the zone; (4) the accepting state is reachable from v. This entails that the maximization operation is not sound.

To achieve this, it is convenient to add an extra clock $\widetilde{z}$ in component A_2 and a fictitious clock 0 which is never reset. Figure 5.1 gives the network. Note that $c_{\max} = 3$. Although clock y does not appear in A_2, one can assume that there are other transitions from q_0 that deal with y. For simplicity, we avoid illustrating these transitions explicitly. Also, when we write $\widetilde{x} = 2$ we mean $\widetilde{x} - 0 = 2$. In the discussion below, v, v' are valuations restricted to $\widetilde{x}, \widetilde{y}, t_1$ and t_2.

In order to reach the state p_2, the synchronization action c needs to be taken: transition sequence $a_1 c$ requires c to be taken at global time 4, and

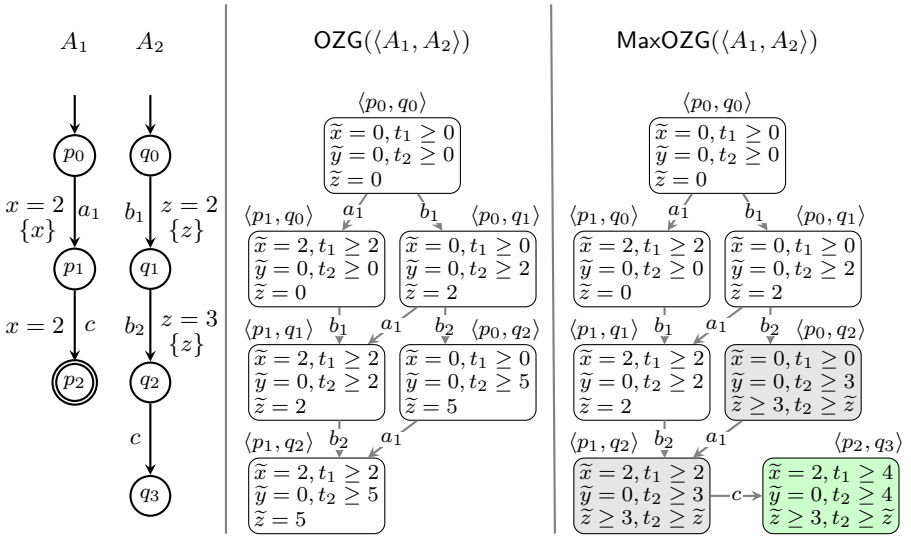


Figure 5.1: Accepting state is not reachable in the network $\langle A_1, A_2 \rangle$, but MaxOZG($\langle A_1, A_2 \rangle$) says otherwise.

transition sequence $b_1 b_2 c$ requires c at global time 5. Hence c is not enabled in the network. This is witnessed by c not being enabled in OZG($\langle A_1, A_2 \rangle$). Valuation v' is present in the zone reached after $b_1 b_2$. The MaxOZG($\langle A_1, A_2 \rangle$) is shown on the right. Zones where maximization makes a difference are shaded gray. In particular, the zone $b_1 b_2$ on maximization adds valuation v, from which $a_1 c$ is enabled, giving a zone in MaxOZG($\langle A_1, A_2 \rangle$) with state p_2. □

5.2. Sync-subsumption for local zones

We have seen that the existing approaches to obtain a finite local zone graph have problems that render them ineffective. From an analysis of these problems, we can conclude that a finite abstraction of the differences between reference clocks constitutes the main challenge. We propose a solution which bypasses the need to worry about such differences: *restrict to synchronized valuations for subsumption.*

We briefly recall the operators sync and global that we introduced in Section 4.5. The sync operator when given a local zone returns the set of synchronized valuations in the local zone, i.e., $\mathsf{sync}(\mathsf{Z}) = \{\mathsf{v} \in \mathsf{Z} \mid \mathsf{v} \text{ is synchronized}\}$. The global operator when applied to a synchronized local valuation v, yields the offset valuation $\overline{v}$ such that $\overline{v}(t) = \mathsf{v}(t_i)$ and $\overline{v}(\widetilde{x}) = \mathsf{v}(\widetilde{x})$ for offset clocks $\widetilde{x} \in \widetilde{X}$. The global operator can be extended to sets of synchronized local

valuations as follows: $\mathsf{global}(\mathsf{sync}(\mathsf{Z})) = \{\overline{v} \mid \overline{v} = \mathsf{global}(\mathsf{v}), \mathsf{v} \in \mathsf{Z}\}$. We have already seen in Lemma 4.9 that $\mathsf{global}(\mathsf{sync}(\mathsf{Z}))$ is a global zone.

Subsumptions over global zones are well studied (we discuss in detail one such subsumption, $\sqsubseteq^{\mathfrak{a}}_{LU}$, in Section 2.8). Taking an off-the-shelf finite abstraction that is based on a simulation $\mathfrak{a}$ and a subsumption relation $\sqsubseteq^{\mathfrak{a}}$ between global zones based on this abstraction, we propose to do the following: given two local zones Z_1 and Z_2, we perform a subsumption test $\mathsf{global}(\mathsf{sync}(\mathsf{Z}_1)) \sqsubseteq^{\mathfrak{a}} \mathsf{global}(\mathsf{sync}(\mathsf{Z}_2))$.

We consider $\mathfrak{a}_{\preccurlyeq LU}$ as this off-the-shelf abstraction. We know that $\mathfrak{a}_{\preccurlyeq LU}$ is finite when considered over global zones (shown in Lemma 2.27). As a consequence, we know that there are only finitely many zones of the form $\mathsf{global}(\mathsf{sync}(\mathsf{Z}))$ that are pairwise incomparable w.r.t. $\mathfrak{a}_{\preccurlyeq LU}$ subsumption.

Remark. *Observe that we could have used any abstraction based on a simulation relation, that is finite when considered over global zones. For instance $\mathfrak{a}_M$ [BBLP06], which is another such abstraction operator, would also have been a suitable candidate.*

Recall that we use the notation $\mathcal{Z} \sqsubseteq^{\mathfrak{a}}_{LU} \mathcal{Z}'$ to indicate that the global zone $\mathcal{Z}$ is subsumed by the global zone $\mathcal{Z}'$. We now define $\sqsubseteq^{\mathfrak{a}LU}_{\mathsf{sync}}$ between local zones.

Definition 5.5. Given two configurations $s := (q, \mathsf{Z})$ and $s' := (q', \mathsf{Z}')$ of the local zone graph, we write $s \sqsubseteq^{\mathfrak{a}LU}_{\mathsf{sync}} s'$ if $q = q'$ and $\mathsf{global}(\mathsf{sync}(\mathsf{Z})) \sqsubseteq^{\mathfrak{a}}_{LU} \mathsf{global}(\mathsf{sync}(\mathsf{Z}'))$.

From the above discussion, it is clear that $\sqsubseteq^{\mathfrak{a}LU}_{\mathsf{sync}}$ is a finite index subsumption relation for local zones. Equipped with such a relation, we can now define *local sync graphs*, that are essentially finite truncations of local zone graphs. We will prove in Theorem 5.2 that local sync graphs are sound and complete w.r.t. reachability.

Definition 5.6 (Local sync graph)**.** A *local sync graph* G of a network of timed automata $\mathcal{N}$ based on $\mathfrak{a}$, is a tree and a subgraph of $\mathsf{LZG}(\mathcal{N})$ satisfying the following conditions:

C0 each node of G is labeled either *covered* or *uncovered*;

C1 the initial node of $\mathsf{LZG}(\mathcal{N})$ belongs to G and is labeled uncovered;

C2 each node of G is reachable from the initial node;

C3 for each uncovered node s, all its successor transitions $s \xrightarrow{a} s'$ occurring in $\mathsf{LZG}(\mathcal{N})$ should be present in G;

C4 for each covered node $s \in G$ there is an uncovered node $s' \in G$ such that $s \sqsubseteq^{\mathfrak{a}LU}_{\mathsf{sync}} s'$. A covered node has no successors.

The above definition essentially translates to this algorithm: explore the local zone graph say in a BFS fashion, and subsume (cover) using $\sqsubseteq^{\mathfrak{a}LU}_{\mathsf{sync}}$. The local sync graph of a network is not unique as it depends on the order of exploration. Using Lemma 2.27, we can conclude that a local sync graph is always finite. Theorem 5.2 below states that local sync graphs are sound and complete for reachability, and this algorithm is correct.

Theorem 5.2 (Soundness and completeness of local sync graphs)**.** *Consider a network of timed automata $\mathcal{N}$, and LU-bounds compatible with guards in $\mathcal{N}$. A state q is reachable in $\mathcal{N}$ iff a node (q, Z), with Z non-empty, is reachable from the initial node in a local sync graph for $\mathcal{N}$ constructed with $\sqsubseteq^{\mathfrak{a}LU}_{\mathsf{sync}}$.*

Proof. If (q, Z) is reachable in a local sync graph then it is trivially reachable in the local zone graph and the (backward) implication follows from soundness of local zone graphs (see Theorem 4.1).

For the other direction which implies completeness of local sync graphs, let us take a global run: $(q_0, \overline{v}_0) \xRightarrow{\delta_1, b_1} (q_1, \overline{v}_1) \cdots \xRightarrow{\delta_n, b_n} (q_n, \overline{v}_n)$. Consider a local sync graph G. By induction on i, for every $(q_i, \overline{v}_i)$ we will find a reachable uncovered node (q_i, Z_i) of G, and a synchronized local valuation $\mathsf{v}_i \in \mathsf{Z}_i$ such that $\overline{v}_i \preceq_{LU} \mathsf{global}(\mathsf{v}_i)$. This proves completeness, since every reachable $(q_i, \overline{v}_i)$ will have a reachable representative node in the local sync graph.

The induction base is immediate, so let us look at the induction step. Consider the global step $(q_i, \overline{v}_i) \xRightarrow{\delta_i, b_i} (q_{i+1}, \overline{v}_{i+1})$. Since $\overline{v}_i \preceq_{LU} \mathsf{global}(\mathsf{v}_i)$, there is a delay δ'_i such that $(q_i, \mathsf{global}(\mathsf{v}_i)) \xRightarrow{\delta'_i, b_i} (q_{i+1}, \overline{v}'_{i+1})$ and $\overline{v}_{i+1} \preceq_{LU} \overline{v}'_{i+1}$. As the global delay δ'_i can be thought of as a sequence of local delays, we have the local run $(q_i, \mathsf{v}_i) \xrightarrow{\Delta, b_i} (q_{i+1}, \mathsf{v}'_{i+1})$, where $\mathsf{v}'_{i+1} = \mathsf{local}(\overline{v}'_{i+1})$. Note that v'_{i+1} is synchronized and $\overline{v}_{i+1} \preceq_{LU} \mathsf{global}(\mathsf{v}'_{i+1})$. From the pre-property of local zones (Lemma 4.6) there exists a transition $(q_i, \mathsf{Z}_i) \xrightarrow{b_i} (q_{i+1}, \mathsf{Z}'_{i+1})$ with $\mathsf{v}'_{i+1} \in \mathsf{Z}'_{i+1}$; in fact, $\mathsf{v}'_{i+1} \in \mathsf{sync}(\mathsf{Z}'_{i+1})$. If $(q_{i+1}, \mathsf{Z}'_{i+1})$ is uncovered, take v'_{i+1} for v_{i+1} and Z'_{i+1} for Z_{i+1} (needed by the induction step). Otherwise, from condition C4, there is an uncovered node $(q_{i+1}, \mathsf{Z}''_{i+1})$ such that $\mathsf{global}(\mathsf{sync}(\mathsf{Z}'_{i+1})) \sqsubseteq^{\mathfrak{a}}_{LU} \mathsf{global}(\mathsf{sync}(\mathsf{Z}''_{i+1}))$. This gives $\mathsf{v}''_{i+1} \in \mathsf{sync}(\mathsf{Z}''_{i+1})$ such that $\mathsf{global}(\mathsf{v}'_{i+1}) \preceq_{LU} \mathsf{global}(\mathsf{v}''_{i+1})$. Now take v''_{i+1} for v_{i+1} and Z''_{i+1} for Z_{i+1}. □

We have already seen in Section 4.5 that the aggregated zone of σ is given by $\mathsf{sync}(\mathsf{Z})$, where (q, Z) is the node reached on σ in the local zone graph. Since we are considering containment with respect to synchronized valuations, it is easy to see that we maintain aggregated zones in local sync graphs as well. Observe that a node being covered implies that further exploration from that node is suspended. Recall that while all the equivalent

sequences of σ lead to the same node in the local zone graph, they could potentially lead to different nodes in the global zone graph. Suppose that there is a covering from the node (q, Z) reached on a sequence σ in the local zone graph. This has the effect of suspending further exploration from all the sequences equivalent to σ. Further, if (q, Z) is covered in the local zone graph, it does not imply that the node $(q, \mathcal{Z})$ of the global zone graph is covered, where $(q, \mathcal{Z})$ is reached on a sequence σ', such that $\sigma' \sim \sigma$. As a consequence, a subsumption in the local zone graph has the effect of stopping the exploration of several executions of the global zone graph. In this sense, sync-subsumption in the local zone graph can be perceived as being "more aggressive" than subsumptions in the global zone graph.

Thus, we have shown that local sync graph, obtained by using sync-subsumption while exploring local zone graphs, is sound and complete with respect to reachability. Furthermore, we have seen that local sync graphs handle interleavings of concurrent actions better than the global zone graph. Thanks to these results, we have an algorithm to test reachability of networks of timed automata that proceeds by exploring the local sync graph.

5.3. Efficient reachability algorithm using local sync graphs

In this section, we present a new reachability algorithm for networks of timed automata that is based on the exploration of local sync graphs. The algorithm takes as input a network of timed automata $\mathcal{N}$, computes its LU-bounds (see Definition 2.37), and constructs a variant of a local sync graph of $\mathcal{N}$, as given in Definition 5.6. An efficient algorithm for sync-subsumption between local zones, i.e., to check if a local zone Z is sync-subsumed by another local zone Z', is given in appendix C. This allows us to store and manipulate local zones while considering the synchronization of these local zones for comparisons; in other words, we do not need to separately store $\mathsf{sync}(\mathsf{Z})$'s or $\mathfrak{a}(\mathsf{sync}(\mathsf{Z}))$'s.

The algorithm proceeds similarly to the standard reachability algorithm, with the main difference that while the standard reachability algorithm constructs and explores the global zone graph and uses the $\sqsubseteq^{\mathfrak{a}}_{LU}$-subsumption between standard zones, our algorithm explores the local zone graph and considers the $\sqsubseteq^{\mathfrak{a}LU}_{\mathsf{sync}}$-subsumption between local zones. Just as the standard algorithm, Algorithm 2 starts the exploration of the local zone graph of $\mathcal{N}$ from the initial node (q_0, Z_0) and maintains two lists of nodes: the list $\mathsf{Visited}$ stores all the nodes constructed by the algorithm, and the list $\mathsf{Waiting}$ stores all the nodes that have been constructed but whose successors have not been computed yet.

Observe that Algorithm 2 removes the covered nodes in lines 8 and 13. Hence, the resulting graph, represented by the set $\mathsf{Visited}$, is not a local sync

Algorithm 2 Reachability algorithm for networks of timed automata based on the exploration of its local sync graph

Input : A network of timed automata $\mathcal{N}$.
Output : true iff $\mathcal{N}$ has a run reaching an accepting state.

1: Set Waiting = Visited := $\{(q_0, \mathsf{Z}_0)\}$
2: **if** q_0 is accepting **then return** true
3: **while** Waiting $\neq \emptyset$ **do**
4: remove some (q, Z) from Waiting
5: **for** all (q', Z') s.t. $(q, \mathsf{Z}) \xrightarrow{a} (q', \mathsf{Z}')$ for some a **do**
6: **if** q' is accepting and $\mathsf{sync}(\mathsf{Z}') \neq \emptyset$ **then**
7: **return** true
8: **else if** $\exists\ (q', \mathsf{Z}'') \in$ Visited s.t. $\mathsf{Z}' \sqsubseteq^{aLU}_{\mathsf{sync}} \mathsf{Z}''$ **then**
9: Skip
10: **else**
11: **for** $(q', \mathsf{Z}'') \in$ Visited **do**
12: **if** $\mathsf{Z}'' \sqsubseteq^{aLU}_{\mathsf{sync}} \mathsf{Z}'$ **then**
13: Remove (q', Z'') from Visited and Waiting
14: Add (q', Z') to Waiting and Visited
15: **return** false

graph according to Definition 5.6. Still, it has all the expected properties that ensure its correctness: every node in Visited is reachable in $\mathsf{LZG}(\mathcal{N})$, and for every reachable node (q, Z) in $\mathsf{LZG}(\mathcal{N})$, there exists a node (q, Z') in Visited such that $(q, \mathsf{Z}) \sqsubseteq^{aLU}_{\mathsf{sync}} (q, \mathsf{Z}')$.

Correctness of the algorithm

We will now prove that Algorithm 2 is correct. The soundness of the algorithm is given by Lemma 5.1, while the completeness follows from Lemma 5.3.

Lemma 5.1. ***(Soundness)*** *If Algorithm 2 returns* true*, then $\mathcal{N}$ has an accepting run.*

Proof. Algorithm 2 returns true by either executing line 2 or line 7. Line 2 is only executed when the initial state q_0 of $\mathcal{N}$ is an accepting state - in this case, the claim is vacuously true. Next, consider the case when line 7 is executed. We can infer from the algorithm that this happens only when a node (q, Z) has just been removed from Waiting, and $(q, \mathsf{Z}) \xrightarrow{a} (q', \mathsf{Z}')$ is such that q' is an accepting state and $\mathsf{sync}(\mathsf{Z}')$ is non-empty. Since (q, Z) was in Waiting, it follows from the algorithm that there is a sequence of actions σ such that $(q_0, \mathsf{Z}_0) \xrightarrow{\sigma} (q, \mathsf{Z})$. This implies that there is a path $(q_0, \mathsf{Z}_0) \xrightarrow{\sigma.a} (q', \mathsf{Z}')$ in $\mathsf{LZG}(\mathcal{N})$. From Lemma 4.6, we know that there is a local run $(q_0, \mathsf{v}_0) \xrightarrow{\sigma.a} (q', \mathsf{v}')$ where $\mathsf{v}' \in \mathsf{Z}'$ is a synchronized local valuation (follows from the fact that $\mathsf{sync}(\mathsf{Z}')$ is non-empty). Then, by Lemma 3.9, we

know that $\mathcal{N}$ has a run $(q_0, \mathsf{global}(\mathsf{v}_0)) \to (q', \mathsf{global}(\mathsf{v}'))$. Since q' is accepting, this is an accepting run of $\mathcal{N}$. □

Lemma 5.2. *Let σ be path in $\mathsf{LZG}(\mathcal{N})$:*

$$(q_0, \mathsf{Z}_0) \xrightarrow{a_1} (q_1, \mathsf{Z}_1) \xrightarrow{a_2} \cdots (q_{n-1}, \mathsf{Z}_{n-1}) \xrightarrow{a_n} (q_n, \mathsf{Z}_n).$$

If Algorithm 2 does not return true *at termination, then for each $0 \leq i \leq n$, there exists a state (q_i, Z'_i) in the set* Visited *such that $\mathsf{Z}_i \sqsubseteq^{\mathfrak{a}LU}_{\mathsf{sync}} \mathsf{Z}'_i$.*

Proof. The proof of the lemma is quite similar to the proof of Lemma 2.32, with the main difference that in this proof, we deal with local zones and $\sqsubseteq^{\mathfrak{a}LU}_{\mathsf{sync}}$ subsumption, in the place of global zones and $\sqsubseteq^{\mathfrak{a}}_{LU}$ subsumption, respectively. □

Lemma 5.3. ***(Completeness)*** *If $\mathsf{LZG}(\mathcal{N})$ has a path to a node (q, Z), where q is accepting and $\mathsf{sync}(\mathsf{Z})$ is non-empty, then Algorithm 2 returns* true*.*

Proof. Suppose that the Algorithm 2 terminated by returning false. Then, by Lemma 5.2, if there is a run to an accepting node (q, Z) of $\mathsf{LZG}(\mathcal{N})$, it follows that (q, Z) is in Visited when the algorithm terminates. But this is not possible because the accepting state is never added to Visited. The only other possibility is that $\mathsf{sync}(\mathsf{Z}) = \emptyset$, but we know that this is not the case. □

From Lemma 5.1 and Lemma 5.3, we can infer that Algorithm 2 is correct. The termination of the algorithm is guaranteed by the finiteness of $\sqsubseteq^{\mathfrak{a}LU}_{\mathsf{sync}}$, which is a direct consequence of the finiteness of $\sqsubseteq^{\mathfrak{a}}_{LU}$ over global zones, given by Lemma 2.27.

An optimization for Algorithm 2

Recall that in the proof of completeness of local sync graph, we consider a local run that sees only synchronized local valuations. Thus, when we construct a path in the local zone graph, we are considering a path passing through $\mathsf{sync}(\mathsf{Z})$'s. Here, we propose an optimization to Algorithm 2 that uses this observation: we ignore exploration from the local zones which contain no synchronized valuations.

First, observe that by restricting exploration from a node, the soundness of the local sync graph is not compromised. Next, we consider the completeness of the restricted local sync graph. Recall that in the proof of completeness of local sync graphs, we consider a global run and trace this run though the nodes of the local zone graph. Observe that in each of the zones encountered in this sequence, there is at least one synchronized valuation, namely the valuation v (or the synchronized valuation v' such that $\mathsf{v} \preceq_{LU} \mathsf{v}'$) seen during the run. Thus, by removing the nodes which has zones with no synchronized

valuations, the feasibility of the required sequence is not affected. As a consequence, the restricted local zone graph is complete.

Consider the restricted local sync graph of a network $\mathcal{N}$ obtained by removing from the local sync graph of $\mathcal{N}$ (given by Definition 5.6) nodes (q, Z) such that $\mathsf{sync}(\mathsf{Z}) = \emptyset$.

Lemma 5.4. *Suppose that* $\mathsf{LZG}(\mathcal{N})$ *has a path to a node* (q, Z)*, where* q *is accepting and* $\mathsf{sync}(\mathsf{Z})$ *is non-empty. Then the restricted local sync graph of* $\mathcal{N}$ *also has path to the node* (q, Z)*.*

Proof. Let $(q_0, \mathsf{Z}_0) \xrightarrow{\sigma} (q, \mathsf{Z})$ and $\mathsf{sync}(\mathsf{Z}) \neq \emptyset$. Since $\mathsf{sync}(\mathsf{Z}) \neq \emptyset$, it follows that there exists a sequence σ', such that $\sigma' \sim \sigma$ and σ' is a global run. From Theorem 4.2, we know that if $(q_0, \mathsf{Z}_0) \xrightarrow{\sigma} (q, \mathsf{Z})$, then $(q_0, \mathsf{Z}_0) \xrightarrow{\sigma'} (q, \mathsf{Z})$. Further, since we know that σ' is a global run, we know that each local zone in the path $(q_0, \mathsf{Z}_0) \xrightarrow{\sigma'} (q, \mathsf{Z})$ contains at least one synchronized valuation, namely the $\mathsf{local}(v)$, where v is a valuation seen in the global run. As a consequence, (q, Z) is reachable in the restricted local sync graph. □

As a consequence of Lemma 5.4, we have the following optimization.

Corollary 5.1 (Optimization)**.** *Let* (q, Z) *be a node of the* $\mathsf{LZG}(\mathcal{N})$ *such that* $\mathsf{sync}(\mathsf{Z}) = \emptyset$*. Then* (q, Z) *need not to be stored, and its successors need not be explored.*

Algorithm 3 presented below, is the resulting updated procedure to check reachability. It is a revised version of Algorithm 2 incorporating the optimization 5.1, implemented by line 8 of the algorithm. Notice that $\mathsf{sync}(\mathsf{Z}_0)$ is not empty since Z_0 contains initial valuations which are by definition synchronized.

5.4. Why local sync graphs are not amenable to POR

In this chapter, we have proposed a new algorithm for testing reachability via exploration of the local sync graph. We have discussed theoretical reasons why the algorithm may be more efficient than the standard reachability algorithm. A natural next step to speed up this algorithm further is to apply a partial order reduction technique while exploring the local sync graph.

However, there is an obstacle to directly applying partial order reduction to local sync graphs. As already discussed, we need to first have an effective way to compute the independence relation between actions in the local sync graph. For local zone graphs, we had showed that if two actions have disjoint domains, then they satisfy the diamond property - in other words, the order in which they are executed does not matter. It was this property that made

Algorithm 3 Reachability algorithm for timed automata based on the exploration of the restricted local sync graph

Input : A network of timed automata $\mathcal{N}$.
Output : true iff $\mathcal{N}$ has a run reaching an accepting state.

```
1: Set Waiting = Visited := {(q_0, Z_0)}
2: if q_0 is accepting then return true
3: while Waiting ≠ ∅ do
4:     remove some (q, Z) from Waiting
5:     for all (q', Z') s.t. (q, Z) -a-> (q', Z') for some a do
6:         if q' is accepting and sync(Z') ≠ ∅ then
7:             return true
8:         else if sync(Z') = ∅ then
9:             Skip
10:        else if ∃ (q', Z'') ∈ Visited s.t. Z' ⊑^aLU_sync Z'' then
11:            Skip
12:        else
13:            for (q', Z'') ∈ Visited do
14:                if Z'' ⊑^aLU_sync Z' then
15:                    Remove (q', Z'') from Visited and Waiting
16:            Add (q', Z') to Waiting and Visited
17: return false
```

local zone graph ideal for applying partial order reduction. It turns out that disjointness of domains does not imply independence of actions in local sync graphs. In particular, in local sync graphs, two actions with disjoint domains violate both the diamond and the forward diamond property. In the next section, we demonstrate this using examples.

We already saw in Lemma 4.8 that two actions with disjoint domains need not satisfy the forward diamond property in the local zone graph. We gave local zone (q, Z) and two actions a and b such that a and b have disjoint domains and are individually enabled from (q, Z), the sequence ab is not enabled from (q, Z). We will now show that two actions with disjoint domains need not satisfy the diamond property either.

$\sqsubseteq^{\mathfrak{a}LU}_{\mathsf{sync}}$ does not preserve the set of enabled actions

We remark that even without forward diamond property it is in principle possible to apply partial-order reduction during exploration. For instance, in the local zone graph, actions with disjoint domains do not satisfy the forward diamond property; yet partial order reduction is feasible in the local zone graph, as actions with disjoint domains satisfy the diamond property in the local zone graph. Had the local zone graph been a finite object, this would have yielded us a working POR procedure for networks of timed automata.

To make the local zone graph finite, we proposed the sync-subsumption between local zones. Unfortunately, sync-subsumption does not preserve the set of enabled actions. In other words, if Z is covered by Z', an action a that is enabled from Z need not be enabled from Z'. As a consequence, actions with disjoint domains do not satisfy the diamond property. We explain this in detail using an example.

Consider the local zone graph given in Figure 5.2. Let Z_2 and Z_2' be as

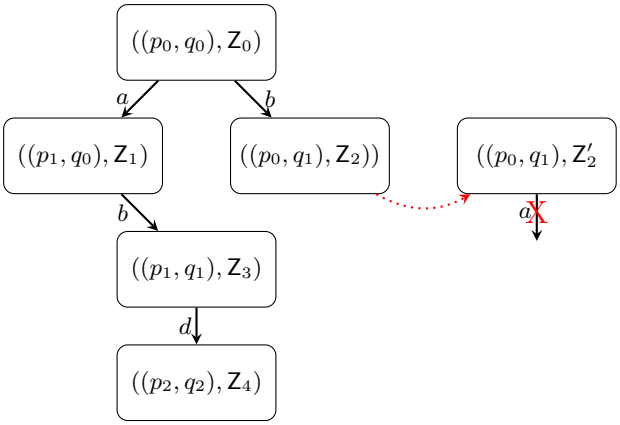


Figure 5.2: An example of a local sync graph to illustrate that local sync graphs are not amenable to partial order reduction techniques. Observe that sync-subsumption does not preserve the set of enabled actions from the zone Z_2. As a consequence, the diamond property does not hold in the local sync graph.

follows and let a be an action with guard $x < 5$.

$$\mathsf{Z}_2 : \widetilde{x} = \widetilde{y} = 0 \wedge t_1 - \widetilde{x} \geq 3 \wedge t_2 - \widetilde{y} \geq 10$$

$$\mathsf{Z}_2' : \widetilde{x} = \widetilde{y} = 0 \wedge t_1 - \widetilde{x} \geq 9 \wedge t_2 - \widetilde{y} \geq 10$$

Clearly,

$$\mathsf{sync}(\mathsf{Z}_2) = \mathsf{sync}(\mathsf{Z}_2') = t_1 = t_2 \wedge t_1 - \widetilde{x} \geq 10 \wedge t_2 - \widetilde{y} \geq 10.$$

It follows that $\mathsf{Z}_2 \sqsubseteq^{aLU}_{\mathsf{sync}} \mathsf{Z}_2'$. However, in this situation a is enabled from Z_2 but not from Z_2'.

This issue primarily arises because the $\sqsubseteq^{aLU}_{\mathsf{sync}}$-inclusion depends only on the synchronized valuations of the two zones, while completely disregarding the non-synchronized valuations. As a consequence, if an action is enabled only from non-synchronized valuations from a local zone Z, it may not be enabled from a local zone Z' that covers Z.

As a consequence, we cannot directly apply a partial order reduction method to the exploration of local sync graph. Consider the local sync graph

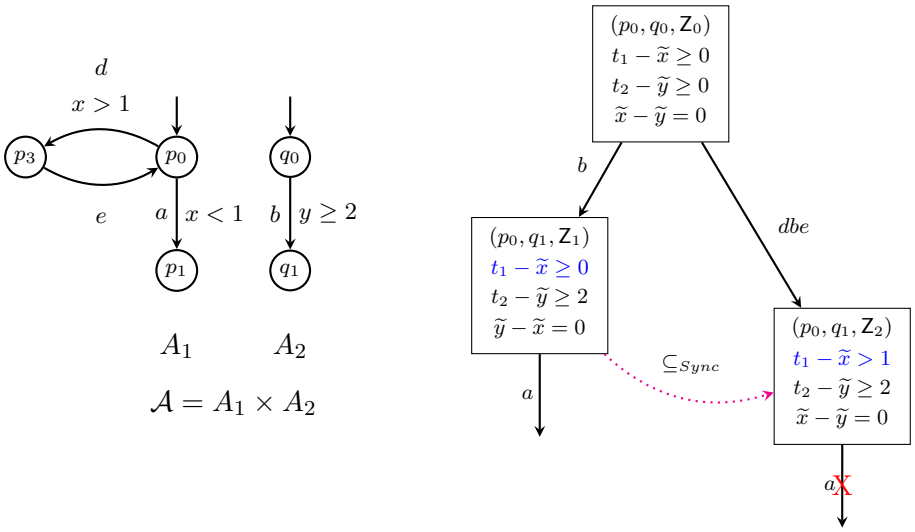


Figure 5.3: An example of two nodes in the local sync graph where sync-subsumption does not preserve the set of enabled actions.

given in Figure 5.2. A partial order reduction algorithm has two (seemingly equivalent) options from the node (p_0, q_0, Z_0) - either to explore the action a or the action b. The algorithm has no preference in this situation. So, an algorithm that chooses exploration of a reaches the node (p_1, q_0, Z_1) from which b is enabled. In this case, the algorithm will report that (p_1, q_1, Z_3) (and eventually (p_2, q_2, Z_4)) is reachable. On the other hand, if the algorithm chooses b, then it reaches the node (p_0, q_1, Z_2) which is covered by the node $(p_0, q_1, \mathsf{Z}'_2)$. Observe that while a was enabled from Z_2, it is not enabled from Z'_2. (We give an example of such a bad situation later in Figure 5.3.)

As a consequence, the algorithm will not report that (p_1, q_1, Z_3) is reachable, which we know is wrong. Thus, if we want to apply partial order reduction to local sync graphs, we need to find a way to account for this phenomenon - perhaps some way to predict when an action may be disabled. Unfortunately, we do not know of a way to do this.

We now give an example of this pathological situation in the local sync graphs. Consider the network $\mathcal{A}$ and its local sync graph given in Figure 5.3. Observe that from the initial node, the path dbe and the action b both lead to nodes of the local zone graph with the same state (p_0, q_1) and the same set of synchronized valuations. As a consequence, if the path dbe is explored before exploring the action b, the node (p_0, q_1, Z_2) reached by b is sync-subsumed by the node (p_0, q_1, Z_1) reached by dbe, as shown in Figure 5.3. Further, we can see that the action a which is enabled from (p_0, q_1, Z_2) is not enabled from (p_0, q_1, Z_1).

Thus, to summarize, in order to make the local zone graph finite, we introduced a subsumption relation $\sqsubseteq^{aLU}_{\mathsf{sync}}$ between local zones. With the

subsumption relation, the exploration of the graph is guaranteed to terminate, but the price to pay was that the diamond property was no longer true. As a consequence, it is not clear how to apply a partial-order reduction procedure on this graph.

If we want to develop a partial order reduction procedure for networks of timed automata, we need to go back to local zone graphs and develop another subsumption technique that renders a transition system for which the independence relation between actions is easy to compute.

Chapter 6

A framework for applying partial order reduction using local zones

In Chapter 5, we proposed a new reachability algorithm for networks of timed automata based on the exploration of local zone graphs. In the algorithm, we compute a transition system called the local sync graph which is a finite truncation of the local zone graph. We discussed theoretical reasons why local sync graphs may be smaller than the standard zone graph. We would now like to apply partial order reduction to the reachability checking procedure based on the exploration of local zone graphs. In this way, we hope to achieve the gains due to both approaches - local time semantics, as well as partial order reduction.

However, we are faced with a major hurdle in this approach - as we pointed out in Section 5.4, we do not know to compute the independence relation between actions in local sync graphs. As a consequence, we do not know how to apply partial order reduction to local sync graph. So, if we want to apply partial order reduction while simultaneously using the local time semantics, we need to develop an alternate subsumption relation between local zones.

From our analysis of the challenges to applying partial order reduction to local sync graphs, we identify the following key problem: $\sqsubseteq^{\mathfrak{a}LU}_{\mathsf{sync}}$ subsumption does not preserve the set of all paths from a node of the local zone graph. An example illustrating this is discussed in Section 6.1. We would like our new subsumption operator to be path-preserving, i.e., if (q, Z) is covered by (q, Z'), then all the paths from (q, Z) are feasible from (q, Z') as well. However, we show that, in general, it is not possible to obtain a finite quotient of the local zone graph that is path-preserving.

Recall that a finite quotient of the standard zone graph was constructed using the $\mathfrak{a}_{\preccurlyeq LU}$ abstraction [BBFL03] that arises from a simulation relation

for global valuations. Proceeding in a similar way, we propose a relation $\equiv^*_M$ between local valuations and prove that $\equiv^*_M$ is a simulation relation (see Definition 2.18). Based on this simulation relation, we then define an abstraction operator $\mathfrak{a}^*_M$ over local zones. It turns out that $\mathfrak{a}^*_M$ is not finite in general. We observe that the main reason for this is the divergence between reference clocks of the different processes. In general, keeping track of arbitrarily large differences between reference clocks appears to be a major inherent difficulty when trying to design finite abstractions for local zone graphs.

We avoid this problem by restricting our attention to those networks, for which, given any feasible sequence of actions in its local zone graph, it is possible to find a run of the network where the differences between reference clocks is bounded at all times. In order to make this idea precise, we introduce the concept of *spread* of a valuation, which is the maximum difference between two reference clocks in the valuation. We say that a run is *D-spread-bounded* if the spread of all the valuations in the run are bounded by a non-negative integer D. A network is *D-spread-bounded* if every local run has an equivalent run that is D-spread-bounded. If a network is D-spread-bounded for some D, we simply say that the network is spread-bounded. Equipped with this notion, we restrict our attention to spread-bounded networks.

Let $\mathcal{N}$ be a network of timed automata that is D-spread-bounded. We show that when computing a covering relation between two nodes of the local zone graph of $\mathcal{N}$, it is sufficient to consider containment with respect to the behaviour of valuations of spread D. We define a subsumption relation $\mathfrak{a}^D_M$ for local zone graphs that does precisely this. $\mathfrak{a}^D_M$ is parameterized by a constant D and compares sets of valuations of spread D. Essentially, this is a generalization of the idea of $\sqsubseteq^{\mathfrak{a}LU}_{\mathsf{sync}}$ introduced in Section 5.2 - while $\sqsubseteq^{\mathfrak{a}LU}_{\mathsf{sync}}$ compares the set of synchronized valuations in two local zones, $\mathfrak{a}^D_M$ compares the set of valuations of spread D. We refer to the transition system obtained by applying the $\mathfrak{a}^D_M$ subsumption to the local zone graph of a network as the $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}$ of the network.

We then show that for a D-spread-bounded network of timed automata, it is sufficient to compute the $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}$ of the network to answer the reachability problem. We remark that the independence relation of $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}$ of a network is easy to compute. By defining the notion of the $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}$ of a D-spread-bounded network, we lay the foundations for applying partial order reduction to these networks. Finally, we motivate the idea of a source-set based partial order reduction method for $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}$ of a D-spread-bounded network. We prove that if $\mathcal{N}$ is D-spread-bounded, then applying the source set to the $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}(\mathcal{N})$ is sound and complete w.r.t. reachability. We make this notion concrete in Chapter 8, where we propose an algorithm that applies partial order reduction to the exploration of the $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}$ of a network of timed automata.

6.1. No finite quotient for local zone graphs

As already shown in Chapter 4, the local zone graph of a network of timed automata is a symbolic representation of the local time semantics of the network that is sound and complete with respect to reachability. As local zone graphs are not finite in general, we need finite abstractions (see Definition 2.35) for local zone graphs to ensure the termination of a reachability procedure based on the exploration of local zone graphs.

We now introduce the notion of an abstract local zone graph, which is a finite representation of a local zone graph. An abstraction operator $\mathfrak{a} : \mathcal{P}(\mathbb{R}_{\geq 0}^{\widetilde{X}'}) \to \mathcal{P}(\mathbb{R}_{\geq 0}^{\widetilde{X}'})$ is a function from sets of local valuations to sets of local valuations.

Definition 6.1 (Abstract local zone graphs)**.** Given a timed automaton A, and an abstraction operator $\mathfrak{a}$, the abstract local zone graph based on $\mathfrak{a}$, denoted as $\mathsf{LZG}^{\mathfrak{a}}(A)$ is a transition system with states of the form (q, W), where q is a state of A and $\mathsf{W} = \mathfrak{a}(\mathsf{W})$ is a set of local valuations. The initial state is (q_0, W_0), where $\mathsf{W}_0 = \mathfrak{a}(\mathsf{Z}_0)$ and Z_0 is the initial zone of $\mathsf{LZG}(A)$, and transitions are of the form $(q, \mathsf{W}) \xrightarrow{t}_{\mathfrak{a}} (q', \mathfrak{a}(\mathsf{W}'))$ if $\mathsf{W} = \mathfrak{a}(\mathsf{W})$ and $\mathsf{W}' = \{\mathsf{v}' \mid \exists\, \mathsf{v} \in \mathsf{W}, \exists\, \delta \in \mathbb{R}_{\geq 0} \text{ s.t. } (q, \mathsf{v}) \xrightarrow{t} \xrightarrow{\delta} (q', \mathsf{v}')\}$ is non-empty.

The abstraction $\mathfrak{a}$ is said to be complete if for any path in $\mathsf{LZG}(A)$ of the form $(q_0, \mathsf{Z}_0) \xrightarrow{\sigma} (q, \mathsf{Z})$, there is a path in $\mathsf{LZG}^{\mathfrak{a}}(A)$ of the form $(q_0, \mathsf{W}_0) \xrightarrow{\sigma}_{\mathfrak{a}} (q, \mathsf{W})$, where $\mathsf{W}_0 = \mathfrak{a}(\mathsf{Z}_0)$. $\mathfrak{a}$ is said to be sound if for any path in $\mathsf{LZG}^{\mathfrak{a}}(A)$ of the form $(q_0, \mathsf{W}_0) \xrightarrow{\rho}_{\mathfrak{a}} (q, \mathsf{W})$ there is a path in $\mathsf{LZG}(A)$ of the form $(q_0, \mathsf{Z}_0) \xrightarrow{\rho} (q, \mathsf{Z})$.

Definition 6.2 (Transition compatible abstraction)**.** Let A be a timed automaton and let $\mathsf{LZG}(A)$ be the local zone graph of A and $\mathsf{LZG}^{\mathfrak{a}}(A)$ be the abstract local zone graph based on an abstraction $\mathfrak{a}$. Let f be a mapping from the nodes of $\mathsf{LZG}(A)$ to the nodes of $\mathsf{LZG}^{\mathfrak{a}}(A)$, such that for all nodes η, η_1, η_2 of $\mathsf{LZG}(A)$

- η and $f(\eta)$ have the same control state.
- if $\eta_1 \xrightarrow{t} \eta_2$ is a transition in $\mathsf{LZG}(A)$, then $f(\eta_1) \xrightarrow{t}_{\mathfrak{a}} f(\eta_2)$ is a transition in $\mathsf{LZG}^{\mathfrak{a}}(A)$.

If such a mapping f exists, then we say that $\mathfrak{a}$ is *transition compatible.*

Lemma 6.1. *If $\mathfrak{a}$ is transition compatible, then $\mathfrak{a}$ is complete.*

Proof. Let $\mathfrak{a}$ be transition compatible and let f be the corresponding mapping from the nodes of $\mathsf{LZG}(A)$ to the nodes of $\mathsf{LZG}^{\mathfrak{a}}(A)$.

Then, it can be observed that if σ is a path in $\mathsf{LZG}(A)$ leading to a node η, there exists a path σ to the node $f(\eta)$ in $\mathsf{LZG}^{\mathfrak{a}}(A)$. By definition of f, the nodes η and $f(\eta)$ have the same control state. □

We will now present a network of timed automata, for which we will show that there exists no finite abstraction that is both sound and transition compatible. The network $\mathcal{A}$ (Figure 6.1) that consists of two processes A_1 and A_2, was originally presented in a similar context in Lugiez et al. [LNZ05]. It is easy to see that any accepting run of the network executes an equal number of a's and b's followed by the global action c.

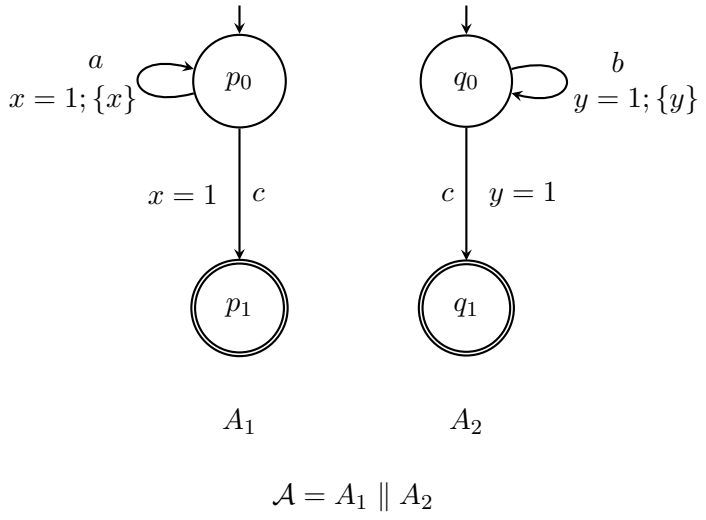


Figure 6.1: A network with no finite quotient for LZG

Consider the local zone graph of $\mathcal{A}$, denoted as $\mathsf{LZG}(\mathcal{A})$. Let $(p_0, q_0, \mathsf{Z}_{i,j})$ be the node of $\mathsf{LZG}(\mathcal{A})$ reached on the execution $a^i b^j$ (let $i > j$) from the initial node:

$$(p_0, q_0, \mathsf{Z}_0) \xrightarrow{a^i b^j} (p_0, q_0, \mathsf{Z}_{i,j})$$

We denote by $\eta_{i,j}$ the node $(p_0, q_0, \mathsf{Z}_{i,j})$ of the local zone graph.

Lemma 6.2. *From the node $\eta_{i,j}$ in the local zone graph of $\mathcal{A}$, the only sequence of the form b^*c reaching the accepting state is $b^{i-j}c$.*

Proof. We know that the network $\mathcal{A}$ accepts the language

$$L_{\mathcal{A}} = \{(ab + ba)^* c\}$$

This is the set of all global runs accepted by $\mathcal{A}$. As a consequence, any local-time run of $\mathcal{A}$ which leads to an accepting state should see an equal number of a's and b's. This follows from the soundness of local time semantics w.r.t. the global semantics.

From the soundness of local zone graphs w.r.t. the local time semantics, we know that any sequence of actions which leads to an accepting node of the local zone graph must have an equal number of a's and b's. Therefore, if

the path ρ leads to an accepting state from $\eta_{i,j}$, since ρ cannot contain a's (by assumption), the number of b's in ρ must be $i-j$, and the number of c's must be 1. Moreover, the action c cannot occur before any occurrence of the action b, since the execution of c disables the action b. The only sequence of actions which satisfies this criterion is $b^{i-j}c$. □

Corollary 6.1. *There are infinitely many distinct nodes $\eta_{i,j}$ in the local zone graph of $\mathcal{A}$.*

Lemma 6.3. *There is no finite abstraction for local zone graphs that is sound and transition compatible.*

Proof. Consider the network $\mathcal{A}$ and its local zone graph $\mathsf{LZG}(\mathcal{A})$. Let $\mathfrak{a}$ be an abstraction of $\mathsf{LZG}(\mathcal{A})$ that is sound and transition compatible. We will show that $\mathfrak{a}$ cannot be finite. Let $\mathsf{LZG}^{\mathfrak{a}}(\mathcal{A})$ denote the abstract local zone graph of $\mathcal{A}$ based on $\mathfrak{a}$. Since $\mathfrak{a}$ is transition compatible, there exists a mapping f from the nodes of $\mathsf{LZG}(\mathcal{A})$ to the nodes of $\mathsf{LZG}^{\mathfrak{a}}(\mathcal{A})$, such that for any node η of $\mathsf{LZG}(\mathcal{A})$, η and $f(\eta)$ have the same control state and if $\eta_1 \xrightarrow{t} \eta_2$ is a transition in $\mathsf{LZG}(\mathcal{A})$, then $f(\eta_1) \xrightarrow{t}_{\mathfrak{a}} f(\eta_2)$ is a transition in $\mathsf{LZG}^{\mathfrak{a}}(\mathcal{A})$.

From Corollary 6.1, we know that there are infinitely many nodes $\eta_{i,j}$ in $\mathsf{LZG}(\mathcal{A})$.

To show that $\mathfrak{a}$ is not finite, we will show that

$$f(\eta_{i,j}) \neq f(\eta_{k,l}) \text{ if } \; i-j \neq k-l$$

Let us assume, for the sake of contradiction, that this is not the case. This implies that there exist two nodes of $\mathsf{LZG}(A)$, $\eta_{i,j}$ and $\eta_{k,l}$, such that $f(\eta_{i,j}) = f(\eta_{k,l})$, and $i-j \neq k-l$.

From Lemma 6.2, we know that from the node $\eta_{i,j}$, the sequence $b^{i-j}c$ leads to an accepting node of $\mathsf{LZG}(\mathcal{A})$. Since $\mathfrak{a}$ is transition compatible, we know that the sequence $b^{i-j}c$ leads to an accepting node from $f(\eta_{i,j})$ in $\mathsf{LZG}^{\mathfrak{a}}(\mathcal{A})$. Similarly, we know that the sequence $b^{k-l}c$ leads to an accepting node from $f(\eta_{k,l})$ in $\mathsf{LZG}^{\mathfrak{a}}(\mathcal{A})$. Since $f(\eta_{i,j}) = f(\eta_{k,l})$, we know that $b^{k-l}c$ leads to an accepting node from $f(\eta_{i,j})$ in $\mathsf{LZG}^{\mathfrak{a}}(\mathcal{A})$.

Since the sequence a^ib^j leads to the node $\eta_{i,j}$ in $\mathsf{LZG}(\mathcal{A})$, we know that a^ib^j leads to $f(\eta_{i,j})$ in $\mathsf{LZG}^{\mathfrak{a}}(\mathcal{A})$. Therefore, we can see that the path $a^ib^jb^{k-l}c$ leads to an accepting node in $\mathsf{LZG}^{\mathfrak{a}}(\mathcal{A})$. Since the abstraction $\mathfrak{a}$ is sound, this means that the run $a^ib^jb^{k-l}c$ leads to an accepting state in $\mathsf{LZG}(\mathcal{A})$. But we know from Lemma 6.2 that this is true only if $i-j = k-l$, which contradicts our assumption. □

The network $\mathcal{A}$ (Figure 6.1) demonstrates a major inherent difficulty when trying to develop finite abstractions for local zone graphs. Recall that for local zone graphs, there is such a finite quotient that is based on $\mathfrak{a}_{\preccurlyeq_{LU}}$ abstraction and $\mathfrak{a}_M$ abstraction [BBFL03, BBLP06].

Observe that the paths posing problems in $\mathsf{LZG}(\mathcal{A})$ are those for which the difference between local times (the values of reference clocks) of two processes grows arbitrary. Our approach will be to restrict our attention to only runs passing only through valuations where the difference between local times is bounded. We say that such a run has bounded spread. The first step is to develop a generalization of $\mathfrak{a}_{\preccurlyeq LU}$ to an abstraction on local zones that preserves all runs of bounded spread. We construct such an abstraction, referred to as $\mathfrak{a}^*_M$, in the next section. We then address the main challenge of using it efficiently to check reachability.

6.2. A region equivalence for local valuations

In Section 2.3, we had presented the notion of regions for global valuations. In this section, we introduce a similar notion of regions for local valuations, referred to as M^*-regions. As in the standard setting, we assume that we are given a bounds function M that determines for each clock x, the maximal constant $M(x)$ used in a guard involving x. We will use a slightly different version of this bounds function which is defined over offset clocks and reference clocks. Our objective is to develop a notion of regions taking into account the bounds function. In contrast to the standard setting, even for a fixed M the number of M^*-regions will be infinite. But we can still define a notion of an abstraction of a local zone similar to $\mathfrak{a}_{\preccurlyeq LU}$ abstraction in the standard setting.

We recall the notation that we use with respect to the set of clocks here. X_p denotes the set of clocks of a process A_p, and $X = \cup_{p \in \mathsf{Proc}} X_p$. Further, we use the following notation for clocks in the offset setting: $\widetilde{X}_p = \{\widetilde{x} \mid x \in X_p\}$, $\widetilde{X}'_p = \widetilde{X}_p \cup \{t_p\}$ and $\widetilde{X}' = \bigcup_p \widetilde{X}'_p$.

6.2.1 M^*-regions

In this section, we introduce a notion of regions for local valuations. We will show that the idea is quite similar to the notion of regions in the setting of offset valuations, discussed in Section 2.3. We first restate a few technical lemmas that we had used to prove results for regions of offset valuations. These are presented with proofs in Section 2.3. We will use these lemmas to prove results about the region equivalence for local valuations.

Lemma 6.4. *For all $x \in \mathbb{R} \setminus \mathbb{Z}$, we have $\{-x\} = 1 - \{x\}$.*

Lemma 6.5. *For $x, y, z \in \mathbb{R}$, $\{z - x\} \leq \{z - y\}$ iff $\lfloor x - y \rfloor = \lfloor x - z \rfloor + \lfloor z - y \rfloor + 1$.*

Lemma 6.6. *For $x, y, z \in \mathbb{R}$, $\{x - z\} \leq \{y - z\}$ iff $\{z - x\} \geq \{z - y\}$.*

Definition 6.3. For local valuations v and v', we define $\mathsf{v} \approx^* \mathsf{v}'$ if for all pairs of offset clock variables (including reference clocks) $x, y \in \widetilde{X}'$, we have $\lfloor \mathsf{v}(x-y) \rfloor = \lfloor \mathsf{v}'(x-y) \rfloor$.

Lemma 6.7. *Let* $\mathsf{v} \approx^* \mathsf{v}'$. *Then, for variables* $x, y, z \in \widetilde{X}'$, *we have* $\{\mathsf{v}(z-x)\} \leq \{\mathsf{v}(z-y)\}$ *iff* $\{\mathsf{v}'(z-x)\} \leq \{\mathsf{v}'(z-y)\}$.

Proof. Follows from Lemma 6.5 and Definition 6.3. □

In Lemma 6.8, we show that local valuations that are $\approx^*$-equivalent can mutually simulate each other w.r.t. the local passage of time. Note that we denote by $\xrightarrow{\delta}_p$ a delay of δ time units in process A_p.

Lemma 6.8. *Let* $\mathsf{v} \approx^* \mathsf{v}'$. *For every local delay* $\mathsf{v} \xrightarrow{\delta}_p \mathsf{u}$, *there exists a* δ' *such that* $\mathsf{v}' \xrightarrow{\delta'}_p \mathsf{u}'$ *where* $\mathsf{u} \approx^* \mathsf{u}'$.

Proof. We assume that $\delta < 1$. If $\delta \geq 1$ then we can decompose it into its integral part and fractional part and repeat the reasoning.

We divide the variable differences into three sets:

$$\begin{aligned} C^+ &= \{t_p - \widetilde{z} \mid z \in X \setminus \{t_p\}\} \\ C^- &= \{\widetilde{z} - t_p \mid z \in X \setminus \{t_p\}\} \\ C^0 &= \{\widetilde{x} - \widetilde{y} \mid x, y \in X \setminus \{t_p\}\} \end{aligned}$$

A local delay of δ increases the value of differences in C^+, decreases the ones in C^- and leaves the C^0 differences unaltered. Consider an element $\phi \in C^+$. Based on the relation between δ and $1 - \{\mathsf{v}(\phi)\}$, its value either stays in the same integer interval, or moves to the next integer point, or to the next integer interval. A symmetric change happens in C^-. We now make this idea more precise.

$$\begin{aligned} \mathsf{u}(t_p - \widetilde{z}) &= \mathsf{v}(t_p - \widetilde{z}) + \delta \\ &= \lfloor \mathsf{v}(t_p - \widetilde{z}) \rfloor + \{\mathsf{v}(t_p - \widetilde{z})\} + \delta \\ &= \lfloor \mathsf{v}(t_p - \widetilde{z}) \rfloor + 1 - \{\mathsf{v}(\widetilde{z} - t_p)\} + \delta \\ \mathsf{u}(\widetilde{z} - t_p) &= \mathsf{v}(\widetilde{z} - t_p) - \delta \\ &= \lfloor \mathsf{v}(\widetilde{z} - t_p) \rfloor + \{\mathsf{v}(\widetilde{z} - t_p)\} - \delta \end{aligned}$$

From the above calculations, we observe two properties:

$$\lfloor \mathsf{u}(t_p - \widetilde{z}) \rfloor = \lfloor \mathsf{v}(t_p - \widetilde{z}) \rfloor + 1 \text{ iff } \delta \geq \{\mathsf{v}(\widetilde{z} - t_p)\} \tag{6.1}$$

$$\lfloor \mathsf{u}(\widetilde{z} - t_p) \rfloor = \lfloor \mathsf{v}(\widetilde{z} - t_p) \rfloor - 1 \text{ iff } \delta > \{\mathsf{v}(\widetilde{z} - t_p)\} \tag{6.2}$$

Note that the difference in the inequalities ($\geq$ in (6.1) and $>$ in (6.2)) is expected, since for any $x \in \mathbb{R}$ we have $\lfloor -x \rfloor = -\lfloor x \rfloor$ if $\{x\} = 0$, and

$\lfloor -x \rfloor = -\lfloor x \rfloor - 1$ otherwise. Among the ordering of fractional parts of differences in C^- for v, consider $(\widetilde{z}_1 - t_p), (\widetilde{z}_2 - t_p)$ that are consecutive in this ordering such that $\{\mathsf{v}(\widetilde{z}_1 - t_p)\} \leq \delta < \{\mathsf{v}(\widetilde{z}_2 - t_p)\}$.

We now propose a δ' as required. From Lemmas 6.6 and 6.7, we know that the fractional parts of differences in C^- are ordered in the same way in v and v'. We take any δ' with $\{\mathsf{v}'(\widetilde{z}_1 - t_p)\} \leq \delta' < \{\mathsf{v}'(\widetilde{z}_2 - t_p)\}$, such that in addition $\delta' = \{\mathsf{v}'(\widetilde{z}_1 - t_p)\}$ if $\delta = \{\mathsf{v}(\widetilde{z}_1 - t_p)\}$. Let $\mathsf{u}' = \mathsf{v}' + \delta'$. Since we started with $\mathsf{v} \approx^* \mathsf{v}'$, from (6.1) and (6.2) we get $u \approx^* \mathsf{u}'$. □

Thanks to these intermediate results, we can now propose a region equivalence for local valuations, for the given clock bounds. As in the standard setting, we consider a bound function M that maps each clock to a non-negative integer. For convenience, we consider a modified function M tailored for clocks in the offset setting. For standard clocks, just as in the standard case, $M(x)$ gives the maximal constant used in a guard involving x. For reference clocks t_p, we set $M(t_p) = \infty$. That is,

$$\begin{aligned} M(\widetilde{x}) &= M(x) && \text{for offset clocks } \widetilde{x} \in \widetilde{X} \\ M(t_p) &= \infty && \text{for reference clocks } t_p \in \widetilde{X} \end{aligned}$$

The notion of M^*-equivalence that we propose ensures that two local valuations can simulate each other in an arbitrary timed automata whose guards respect M bounds.

Definition 6.4 (M^*-equivalence)**.** Let $M : \bigcup_{p \in Proc} X_p \cup \{t_p\} \mapsto \mathbb{N} \cup \{\infty, -\infty\}$ be a *bounds function* mapping each clock to a non-negative constant, $-\infty$ or ∞. For a local valuation v, let

$$\mathsf{Bounded}(\mathsf{v}) = \bigcup_{p \in Proc} \{x \in \widetilde{X}'_p \mid \mathsf{v}(t_p - x) \leq M(x))\}$$

Note that by definition, all the reference clocks belong to $\mathsf{Bounded}(\mathsf{v})$.

Two local valuations are M^*-*equivalent*, denoted as $\mathsf{v} \equiv^*_M \mathsf{v}'$, if

- $\mathsf{Bounded}(\mathsf{v}) = \mathsf{Bounded}(\mathsf{v}')$
- $\mathsf{v}_{\restriction B} \approx^* \mathsf{v}'_{\restriction B}$ where $\mathsf{v}_{\restriction B}$ and $\mathsf{v}'_{\restriction B}$ denote the valuations v and v' restricted to clocks in $B = \mathsf{Bounded}(\mathsf{v})$.

We write $[\mathsf{v}]^{M^*}$ to denote the equivalence class of v under $\equiv^*_M$ and refer to it as the M^*-*region* of v.

For example, consider the network $\mathcal{A}$ (Figure 6.1), that contains two processes A_1 and A_2. Let t_1, t_2 be the reference clocks of A_1, A_2 respectively. From the network, we have $M(\widetilde{x}) = M(\widetilde{y}) = 1, M(t_1) = M(t_2) = \infty$. A local region for $\mathcal{N}$ is given by one of the four possible intervals

$[0,0],(0,1),[1,1],(1,\infty)$ for the differences $t_1 - \widetilde{x}$, $t_2 - \widetilde{y}$ and $\widetilde{x} - \widetilde{y}$ and one of the infinitely many intervals $[0,0],(0,1),[1,1],(1,2),[2,2],\ldots$ for (the modulus of) $t_1 - t_2$. This is because by Definition 6.4, all reference clocks t_p are automatically in $\mathsf{Bounded}(\mathsf{v})$ and the exact integral values between their differences need to be maintained. As a consequence, there are infinitely many M^*-regions. The example of network $\mathcal{A}$ (Figure 6.1) shows that this is the price to pay for working with local valuations.

We say that a network of timed automata *conforms to bounds function* M if for each process p and clock $x \in X_p$, all the guards testing x use constants not bigger than $M(x)$. We write $(q,\mathsf{v}) \equiv^*_M (q,\mathsf{v}')$ when $\mathsf{v} \equiv^*_M \mathsf{v}'$.

Recall the definition of a simulation relation given in Definition 2.18. A simulation relation on the local time semantics is a binary relation between configurations $(q,\mathsf{v}) \preccurlyeq (q,\mathsf{v}')$ such that:

- $\preccurlyeq$ is reflexive and transitive,
- for every local delay-action sequence $(q,\mathsf{v}) \xrightarrow{\Delta,b} (q_1,v_1)$, there exists a local delay sequence Δ' such that $(q,\mathsf{v}') \xrightarrow{\Delta',b} (q_1,\mathsf{v}'_1)$ and $(q_1,v_1) \preccurlyeq (q_1,\mathsf{v}'_1)$.

The relation $\preccurlyeq$ is a bisimulation if $\preccurlyeq$ is also symmetric.

Proposition 6.1. *Let M be a bounds function. Consider a network of timed automata conforming to M. The relation $(q,\mathsf{v}) \equiv^*_M (q,\mathsf{v}')$ is a bisimulation.*

Proof. From Lemma 6.8, we know that for each local delay transition $(q,\mathsf{v}) \xrightarrow{\delta}_p (q,\mathsf{u})$ there is a local delay $(q,\mathsf{v}') \xrightarrow{\delta'}_p (q,\mathsf{u}')$ such that $\mathsf{u} \equiv^*_M \mathsf{u}'$.

Next, we consider the case of action transitions. Since $\mathsf{v} \equiv^*_M \mathsf{v}'$, we have $\mathsf{v}(t_p - \widetilde{x}_p) > M(x_p)$ iff $\mathsf{v}'(t_p - \widetilde{x}_p) > M(x_p)$, for all $x_p \in X_p$ and $p \in \mathsf{Proc}$. Moreover, when x_p is bounded, $\lfloor \mathsf{v}(t_p - \widetilde{x}_p) \rfloor = \lfloor \mathsf{v}'(t_p - \widetilde{x}_p) \rfloor$ and $\lfloor \mathsf{v}(\widetilde{x}_p - t_p) \rfloor = \lfloor \mathsf{v}'(\widetilde{x}_p - t_p) \rfloor$. Hence, v satisfies a guard $x_p \sim c$ with $\sim \in \{<,\leq,>,\geq\}$ and $c \leq M(x_p)$ iff v' satisfies the guard. Further, as a reset of x_p results in $\mathsf{v}(\widetilde{x}_p) = \mathsf{v}(t_p)$, it is clear that if $\mathsf{v} \equiv^*_M \mathsf{v}'$, then $[R]\mathsf{v} \equiv^*_M [R]\mathsf{v}'$. □

From Proposition 6.1, we infer that $[\mathsf{v}]^{M^*}$ is a region analogous to the standard case, for the setting of local valuations.

6.2.2 Abstraction operation $\mathfrak{a}^*_M$

Equipped with the notion of M^*-regions we can define the operation of $\mathfrak{a}^*_M$-abstraction for local zones. In the standard setting, $\mathfrak{a}_{\preccurlyeq LU}$ abstraction of a zone is a union of all LU-regions intersecting the zone. Analogously, we define $\mathfrak{a}^*_M$ abstraction of a local zone to be the union of all M^*-regions intersecting the local zone.

Definition 6.5 ($\mathfrak{a}^*_M$ abstraction)**.** For a local zone Z, we define $\mathfrak{a}^*_M(\mathsf{Z}) := \{\mathsf{v} \mid \exists\, \mathsf{v}' \in \mathsf{Z} \text{ with } \mathsf{v} \equiv^*_M \mathsf{v}'\}$.

Recall that in the standard setting, the abstraction of a zone is not a zone. Similarly, in our setting, the abstraction of a local zone may not be a local zone. In Section 6.4, we will show that in spite of this, the non-inclusion test $\mathsf{Z} \not\subseteq \mathfrak{a}^*_M(\mathsf{Z}')$ can be done efficiently.

Observe that $\mathfrak{a}^*_M$ abstraction is not finite in general. This is a direct consequence of the existence of infinitely many M^*-regions. For instance, in the local zone graph of $\mathcal{A}$ (Figure 6.1), the local zone Z_{ij} reached on the execution $a^i b^j$ has the constraints: $t_1 - \widetilde{x} \geq i \wedge t_2 - \widetilde{y} \geq j \wedge \widetilde{x} - \widetilde{y} = i - j$. Pick a valuation v_{ij} with: $t_1 - \widetilde{x} = 1 \wedge t_2 - \widetilde{y} = 1 \wedge t_1 - t_2 = i - j \wedge \widetilde{x} - \widetilde{y} = i - j$. From the constraints in zone Z_{ij}, we see that for a pair i', j' different from i, j we have $\mathsf{v}_{ij} \notin \mathsf{Z}_{i'j'}$ due to the $\widetilde{x} - \widetilde{y}$ constraint. Hence for two different zones of the form Z_{ij}, it cannot be the case that one subsumes the other. This gives an infinite local zone graph, even with the $\mathfrak{a}^*_M$ abstraction. In Section 6.3, we will examine situations when the $\mathfrak{a}^*_M$ abstraction becomes finite.

6.3. Bounding the spread

In Section 6.2, we proposed a relation $\equiv^*_M$ between valuations in the local time semantics, and we proved that $\equiv^*_M$ is a simulation relation for local valuations. Further, we showed that the abstraction relation $\mathfrak{a}^*_M$ constructed using this simulation is not finite. We observe that the key issue that prevents us from obtaining finite abstractions of local zone graphs is the lack of bound on the differences between reference clocks.

As a solution to this problem, we restrict our attention to those networks for which, given any feasible sequence of actions in its local zone graph, it is possible to find a run where the difference between reference clocks is always bounded. To make this idea precise, we introduce the notion of the *spread* of a valuation, which is defined as the maximum difference between two reference clocks in the valuation. We say a run is *spread-bounded* if the spread of all valuations in the run is bounded by some non-negative integer D. Equipped with this notion, we restrict our attention to those networks in which each run has an equivalent spread-bounded run. For local zone graphs of networks of timed automata that are D-spread-bounded, we show that it is sufficient to look at valuations of spread D, in order to get a finite local zone graph, that is amenable to partial order reduction.

We will now formally define the notion of spread and spread-boundedness.

Definition 6.6. The *spread* between processes A_p, A_q in a valuation v is the absolute value of the difference between their reference clocks: $|\mathsf{v}(t_p) - \mathsf{v}(t_q)|$.

We say that a *valuation* v *has spread* D if the spread between every pair of processes in v is at most D.

Definition 6.7. Consider a run in the local time semantics

$$(q_0, \mathsf{v}_0) \xrightarrow{\Delta_1} \xrightarrow{a_1} (q_1, \mathsf{v}_1) \xrightarrow{\Delta_2} \xrightarrow{a_2} \cdots (q_{n-1}, \mathsf{v}_{n-1}) \xrightarrow{\Delta_n} \xrightarrow{a_n} (q_n, \mathsf{v}_n)$$

We say that it is *D-spread* if all $\mathsf{v}_0, \ldots, \mathsf{v}_n$ have spread D.

We now present examples of D-spread runs of a network. Consider the network $\mathcal{A}$ given in Figure 6.1. Observe that between the execution of two a actions, the process A_1 needs to elapse exactly one unit of time. Likewise, between the execution of two b actions, the process A_2 needs to elapse exactly one unit of time.

Consider the following run of $\mathcal{A}$.

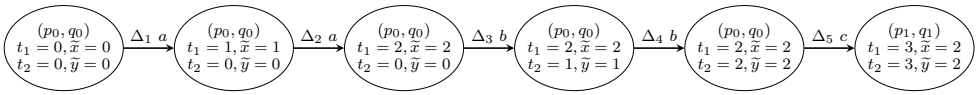


Figure 6.2: A run of $\mathcal{A}$ of spread 2

From inspection of the run given in Figure 6.2, we see that the maximum spread between A_1 and A_2 is in v_2, and $\mathsf{v}_2(t_1 - t_2) = 2$. Thus, the spread between A_1 and A_2 in v_2 is 2 and as a consequence, the run is 2-spread.

Consider another run of $\mathcal{A}$ given in Figure 6.3.

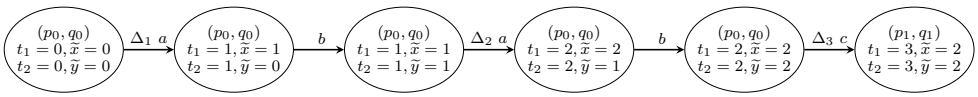


Figure 6.3: A run of $\mathcal{A}$ of spread 0

From Figure 6.3, we see that for all valuations in the run, the spread between A_1 and A_2 is 0, and therefore, the run is 0-spread.

Next, we define when a network of timed automata is said to be spread-bounded.

Definition 6.8. A network of timed automata $\mathcal{N}$ is said to be *D-spread-bounded* if every local run of $\mathcal{N}$ can be converted to a D-spread run by adjusting the delays.

Observe that if a network $\mathcal{N}$ is D_1-spread, from Definition 6.8, it follows that $\mathcal{N}$ is D_2-spread, for all $D_2 \geq D_1$. On a different note, we remark that in order to convert a run to a D-spread run, we can adjust also the delays of processes that do not participate in the next action.

Just as we defined $\mathsf{sync}(\mathsf{Z})$ as the restriction of the local zone Z to synchronized local valuations (see Section 5.2), we define $\mathsf{spread}_D(\mathsf{Z})$ as the

restriction of Z to local valuations of spread D. This allows us to define a finite abstraction operation, and a subsumption operation based on it.

Definition 6.9 ($\mathfrak{a}_M^D$-subsumption)**.** Let D be an integer and M a bounds function. For a zone Z we define:

$$\mathsf{spread}_D(\mathsf{Z}) = \mathsf{Z} \cap \{t_p - t_q \leq D : p, q \in \mathsf{Proc}\} \ .$$

We define the $\mathfrak{a}_M^D$ *subsumption operation*, denoted as $\mathsf{Z} \sqsubseteq_M^D \mathsf{Z}'$, by

$$\mathsf{spread}_D(\mathsf{Z}) \subseteq \mathfrak{a}_M^*(\mathsf{spread}_D(\mathsf{Z}')) \ .$$

Definition 6.10 ($\mathsf{LZG}_{\mathsf{M}}^{\mathsf{D}}(\mathcal{N})$)**.** Let $\mathcal{N}$ be a network of timed automata conforming to a bound function M. A *local zone graph with* $\mathfrak{a}_M^D$*-subsumption* for $\mathcal{N}$, denoted $\mathsf{LZG}_{\mathsf{M}}^{\mathsf{D}}(\mathcal{N})$, is a graph whose nodes are pairs (q, Z) where q of a state of $\mathcal{N}$ and Z is a local zone. The initial node of $\mathsf{LZG}_{\mathsf{M}}^{\mathsf{D}}(\mathcal{N})$ is the initial node (q_0, Z_0) of $\mathsf{LZG}(\mathcal{N})$ and for each node (q, Z) either:

- the node is *uncovered*, namely, for each transition $(q, \mathsf{Z}) \xrightarrow{a} (q', \mathsf{Z}')$ in $\mathsf{LZG}(\mathcal{N})$ there is transition $(q, \mathsf{Z}) \xrightarrow{a} (q', \mathsf{Z}')$ in $\mathsf{LZG}_{\mathsf{M}}^{\mathsf{D}}(\mathcal{N})$ if $\mathsf{spread}_D(\mathsf{Z}')$ is not empty; or
- the node is *covered*, namely, (q, Z) has no outgoing transitions, and there exists an uncovered node (q, Z') such that $\mathsf{Z} \sqsubseteq_M^D \mathsf{Z}'$.

Note that the definition implies that the initial node is uncovered.

In Theorem 4.1, we reduced the reachability problem for $\mathcal{N}$ to reachability in $\mathsf{LZG}(\mathcal{N})$. More precisely, $\mathcal{N}$ has a run executing a chosen action α iff there is a path in $\mathsf{LZG}(\mathcal{N})$ starting from the initial node and containing the α edge. The next sequence of lemmas implies that $\mathsf{LZG}_{\mathsf{M}}^{\mathsf{D}}(\mathcal{N})$ is sound and complete w.r.t. reachability. In other words, we will show that $\mathsf{LZG}_{\mathsf{M}}^{\mathsf{D}}(\mathcal{N})$ can be used in place of $\mathsf{LZG}(\mathcal{N})$. Moreover, $\mathsf{LZG}_{\mathsf{M}}^{\mathsf{D}}(\mathcal{N})$ is a finite abstraction of $\mathsf{LZG}(\mathcal{N})$. Additionally, if $\mathcal{N}$ is D-spread-bounded then we can use a partial order reduction method while exploring it.

Remark. *Until now, we always defined acceptance as reaching a state of the network. We show that it is possible to convert a standard network that accepts by reaching an accepting state to a network that accepts by executing an action at the end. A detailed discussion of this transformation is presented in Appendix B.*

Lemma 6.9. *For every D and a bounds function M that does not map any offset clock to ∞, the graph $\mathsf{LZG}_{\mathsf{M}}^{\mathsf{D}}(\mathcal{N})$ is finite.*

Proof. First, observe that there are only finitely many sets of the form $\mathfrak{a}_M^*(\mathsf{spread}_D(\mathsf{Z}))$ for some zone Z. Indeed, such a set is a collection of M^*-regions of local valuations belonging to $\mathsf{spread}_D(\mathsf{Z})$. We will show that there

is a bound on the constants in constraints defining $[\mathsf{v}]^{M^*}$ (cf. Definition 6.4) for $\mathsf{v} \in \mathsf{spread}_D(\mathsf{Z})$. As the spread of v is bounded by D, the difference between each pair of reference clocks t_p and t_q is bounded by D. If a clock x is unbounded then there are no constraints between x and other variables. If x is a clock of A_p, and x is bounded then $t_p - \widetilde{x}_p < M(\widetilde{x}_p)$. This implies that $t_q - \widetilde{x}_p < M(\widetilde{x}_p) + D$ for all other reference clocks t_q. For the constraint $\widetilde{x} - t_p < c$, we know that $c \leq 0$ since $\widetilde{x} \leq t_p$. Then, it follows that $-c \leq M(\widetilde{x}_p)$, as otherwise $\widetilde{x} - t_p < -M(\widetilde{x}_p)$ giving $t_p - \widetilde{x} > M(\widetilde{x}_p)$, which is inconsistent with $t_p - \widetilde{x}_p < M(\widetilde{x}_p)$. Lastly, for a constraint of the form $\widetilde{x} - \widetilde{y} < c$, we can deduce, for the same reasons that $c \leq D + \max(M(\widetilde{x}), M(\widetilde{y}))$. □

Having defined $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}$ and proved its finiteness, we now state a property of $\sqsubseteq^D_M$ subsumption, which will be useful in proving results about $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}$ later.

Lemma 6.10. *Suppose that $\mathcal{N}$ is a D-spread-bounded system conforming to the bounds function M and (q, Z) is a node reachable in $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}(\mathcal{N})$. Then, if $\mathsf{Z} \sqsubseteq^D_M \mathsf{Z}'$ and $(q, \mathsf{Z}) \xrightarrow{a} (q_1, \mathsf{Z}_1)$, then $(q, \mathsf{Z}') \xrightarrow{a} (q_1, \mathsf{Z}'_1)$ for some Z'_1 with $\mathsf{Z}_1 \sqsubseteq^D_M \mathsf{Z}'_1$.*

Proof. Since $(q, \mathsf{Z}) \xrightarrow{a} (q_1, \mathsf{Z}_1)$, by the pre-post property of local zones, for each $\mathsf{v}_1 \in \mathsf{Z}_1$, we have $\mathsf{v} \in \mathsf{Z}$ such that $(q, \mathsf{v}) \xrightarrow{a} (q_1, \mathsf{v}_1)$. Consider v_1 that is D-spread-bounded. Since $\mathcal{N}$ is D-spread-bounded, every local run of $\mathcal{N}$ can be converted to a D-spread run. As a consequence, we can assume that v is D-spread-bounded. By definition of $\mathfrak{a}^D_M$ subsumption, there is a $\mathsf{v}' \in \mathsf{Z}'$ such that $\mathsf{v} \equiv^*_M \mathsf{v}'$. Since v is D-spread-bounded, so is v', by definition of $\equiv^*_M$. Further, from Proposition 6.1, we also have $(q, \mathsf{v}') \xrightarrow{a} (q_1, \mathsf{v}'_1)$ such that $\mathsf{v}_1 \equiv^*_M \mathsf{v}'_1$. □

Since $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}$ is obtained from LZG by removing some transitions, we immediately obtain Lemma 6.11.

Lemma 6.11. *For each action α, if action α is reachable by a path in $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}(\mathcal{N})$, then it is also reachable by a path in $\mathsf{LZG}(\mathcal{N})$.*

Now, it remains to show that $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}$ is complete. This is given by Lemma 6.12. The proof follows similarly to the proof of completeness of local sync graphs (Theorem 5.2).

Lemma 6.12. *Given a network of timed automata $\mathcal{N}$, consider M-bounds function compatible with guards in $\mathcal{N}$. If q is reachable in $\mathcal{N}$, then in $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}(\mathcal{N})$ a node (q, Z) is reachable from the initial node, for some non-empty zone Z.*

Proof. Consider a global run of $\mathcal{N}$: $(q_0, \overline{v}_0) \xrightarrow{\delta_1, b_1} (q_1, \overline{v}_1) \cdots \xrightarrow{\delta_n, b_n} (q_n, \overline{v}_n)$. By induction on i, for every $(q_i, \overline{v}_i)$ we can find a reachable uncovered node

(q_i, Z_i) of $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}(\mathcal{N})$, and a local (in fact, synchronized) valuation v_i, such that $\mathsf{v}_i \in \mathsf{Z}_i$ and $\mathsf{local}(\overline{v}_i) \equiv^*_M \mathsf{v}_i$. Since a synchronized valuation has spread D by definition, this establishes the completeness of $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}$, since every reachable $(q_i, \overline{v}_i)$ will have a reachable representative node in $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}(\mathcal{N})$.

The argument is quite similar to the proof of Theorem 5.2, with the difference that in this proof, we consider $\mathfrak{a}^*_M$ subsumption instead of $\mathfrak{a}_{\preccurlyeq LU}$ subsumption. The base case of the induction follows directly. Next, we look at the induction step. Consider the global step $(q_i, \overline{v}_i) \xRightarrow{\delta_i, b_i} (q_{i+1}, \overline{v}_{i+1})$. Since the global delay δ_i can be thought of as a sequence of local delays, we have the local run $(q_i, \mathsf{local}(\overline{v}_i)) \xrightarrow{\Delta_i, b_i} (q_{i+1}, \mathsf{local}(\overline{v}_{i+1}))$. By induction hypothesis, we have $\mathsf{v}_i \in \mathsf{Z}$ such that $\mathsf{v}_i \equiv^*_M \mathsf{local}(\overline{v}_i)$. From Lemma 6.1, we know that there exists a delay Δ'_i such that $(q_i, \mathsf{v}_i) \xrightarrow{\Delta'_i, b_i} (q_{i+1}, \mathsf{v}_{i+1})$ such that $\mathsf{v}_{i+1} \equiv^*_M \mathsf{local}(\overline{v}_{i+1})$. Since $\mathsf{local}(\overline{v}_{i+1})$ is a synchronized valuation, from the definition of $\equiv^*_M$, it follows that v_{i+1} is also synchronized. Using the pre-property of local zones (Lemma 4.6), there exists a transition $(q_i, \mathsf{Z}_i) \xrightarrow{b_i} (q_{i+1}, \mathsf{Z}_{i+1})$ with $\mathsf{v}_{i+1} \in \mathsf{Z}_{i+1}$ (in fact, $\mathsf{v}_{i+1} \in \mathsf{sync}(\mathsf{Z}_{i+1})$). If $(q_{i+1}, \mathsf{Z}_{i+1})$ is uncovered, then v_{i+1} and Z_{i+1} are as needed for the induction step. Otherwise, by the definition of $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}$ (Definition 6.10), there exists an uncovered node $(q_{i+1}, \mathsf{Z}''_{i+1})$ such that $\mathsf{Z}'_{i+1} \sqsubseteq^D_M \mathsf{Z}''_{i+1}$. Since $\mathsf{v}_{i+1} \in \mathsf{spread}_D(\mathsf{Z}_{i+1})$, we have $\mathsf{v}'_{i+1} \in \mathsf{spread}_D(\mathsf{Z}'_{i+1})$ such that $\mathsf{v}_{i+1} \equiv^*_M \mathsf{v}'_{i+1}$. In this case, we take v'_{i+1} for v_{i+1} and Z'_{i+1} for Z_{i+1}. □

We now present an alternate way to obtain the completeness for $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}$. Observe that the $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}$ with $D = 0$, is the graph we would get if we use $\mathfrak{a}_M$ subsumption instead of $\mathfrak{a}_{\preccurlyeq LU}$ subsumption, while computing the local sync graph. Further, observe that using $\mathfrak{a}^*_M$ subsumption instead of $\mathfrak{a}_{\preccurlyeq LU}$ subsumption does not affect the correctness of local sync graph. Then, from the completeness of local sync graphs, we immediately get the completeness of $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}$ with $D = 0$, for all networks. Since increasing the parameter spread D while computing $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}$ maintains the reachable nodes of $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}$ with lower values of D, this does not affect the completeness.

6.3.1 Partial order reduction for $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}$

As promised, in this section, we will show how it is possible to use partial order exploration in $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}$. We will adopt a quite abstract view of partial order reduction. Later we will explain our choice by showing that it would not make sense to refer to diamonds in $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}$ when defining a reduction.

A partial order reduction can be seen as a way of defining a function $\mathsf{source} : Q \to \mathcal{P}(\Sigma)$ assigning a set of actions to each state of $\mathcal{N}$. A *source transition* in LZG is a transition $(q, \mathsf{Z}) \xrightarrow{a} (q', \mathsf{Z}')$ in LZG such that $a \in \mathsf{source}(q)$. Observe, that source depends only on the state component of the configuration. A *source path* is a path consisting of only source transitions.

A source function is *complete* for a LZG, if whenever there is a path executing α then there is a source path executing α. Note that if α is not reachable then every source function is complete. If α is reachable, and we are given a path to α then we can take a source function that chooses exactly this path. Our goal is to have a source function that is easier to calculate than solving the reachability problem in the first place.

We define a *path with subsumption edges* as a path of $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}$ in which edges can either edges of the LZG or subsumption edges between nodes of the LZG. Now, we can also talk about source paths with subsumption edges.

Lemma 6.13. *Suppose that $\mathcal{N}$ is a D-spread-bounded system conforming to the bounds function M. For every path $(q_0, \mathsf{Z}_0) \xrightarrow{a_1} (q_1, \mathsf{Z}_1) \xrightarrow{a_2} \dots (q_n, \mathsf{Z}_n)$ in $\mathsf{LZG}(\mathcal{N})$, there exists a path with subsumption edges in $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}(\mathcal{N})$ $(q_0, \mathsf{Z}_0) \xrightarrow{a_1} (q_1, \mathsf{Z}'_1) \xrightarrow{a_2} \dots (q_n, \mathsf{Z}'_n)$, such that $\mathsf{spread}_D(\mathsf{Z}_n) \subseteq \mathfrak{a}^*_M(\mathsf{spread}_D(\mathsf{Z}'_n))$. If the initial path is a source path, so is the path in $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}$.*

Proof. Consider a path $\sigma = a_1 a_2 \cdots a_n$ in $\mathsf{LZG}(\mathcal{N})$. Consider a local run that is an instantiation of this path: $(q_0, \mathsf{v}_0) \xrightarrow{a_1} (q_1, \mathsf{v}_1) \xrightarrow{a_2} \dots \xrightarrow{a_n} (q_n, \mathsf{v}_n)$. Since $\mathcal{N}$ is D-spread-bounded, we may assume that each valuation in this run has spread D. We will show, by induction on i, that the path $a_1 a_2 \cdots a_n$ exists in $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}(\mathcal{N})$. From our induction hypothesis, we know that there is a path till (q_i, Z'_i) and that there is a D-spread-bounded valuation $\mathsf{v}'_i \in \mathsf{Z}'_i$, with a path $(q_i, \mathsf{v}'_i) \xrightarrow{a_{i+1}} (q_{i+1}, \mathsf{v}'_{i+1}) \xrightarrow{a_{i+2}} \cdots \xrightarrow{a_n} (q_n, \mathsf{v}'_n)$ where all valuations are D-spread-bounded. Clearly, the induction hypothesis holds for the initial configuration. For the induction step we look at the definition of $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}$.

If (q_i, Z'_i) is not covered, then we have $(q_i, \mathsf{Z}'_i) \xrightarrow{a_{i+1}} (q_{i+1}, \mathsf{Z}'_{i+1})$ with $\mathsf{v}'_{i+1} \in \mathsf{Z}'_{i+1}$. In this case, we are done with the induction step.

Next, consider the case where (q_i, Z'_i) is covered. In this case, there is an uncovered node (q_i, Z''_i) with $\mathsf{Z}'_i \sqsubseteq^D_M \mathsf{Z}''_i$. By definition of $\mathfrak{a}^D_M$ subsumption, there is a $\mathsf{v}''_i \in \mathsf{Z}''_i$ such that $\mathsf{v}'_i \equiv^*_M \mathsf{v}''_i$. Since v'_i is D-spread-bounded, by definition of $\equiv^*_M$, we know that v''_i is also D-spread-bounded. We also have a run $(q_i, \mathsf{v}''_i) \xrightarrow{a_{i+1}} (q_{i+1}, \mathsf{v}''_{i+1}) \xrightarrow{a_{i+2}} \dots \xrightarrow{a_n} (q_n, \mathsf{v}''_n)$ by Proposition 6.1. Since $\mathcal{N}$ is D-spread-bounded, we can assume that this run is D-spread-bounded. Thus, we are done with the induction step in this case also.

Finally, since source depends only on the state component, the above path is a source path if the initial one was. □

From Lemma 6.11 and Lemma 6.13, we immediately get Theorem 6.1.

Theorem 6.1. *Suppose that $\mathcal{N}$ is a D-spread-bounded system conforming to M. If* source *is a complete function for $\mathsf{LZG}(\mathcal{N})$ then α is reachable on a run of $\mathcal{N}$ iff it is reachable on a source path of $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}(\mathcal{N})$.*

The formulation given above with source sets may seem more abstract than necessary - we will now explain why this is useful. We first examine

more closely whether the diamond and the forward diamond properties (see Definition 2.48) hold in $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}(\mathcal{N})$ for actions with disjoint domains. We show that, in general, actions with disjoint domains do not satisfy the diamond property in $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}(\mathcal{N})$. We illustrate this schematically using an example in Figure 6.4.

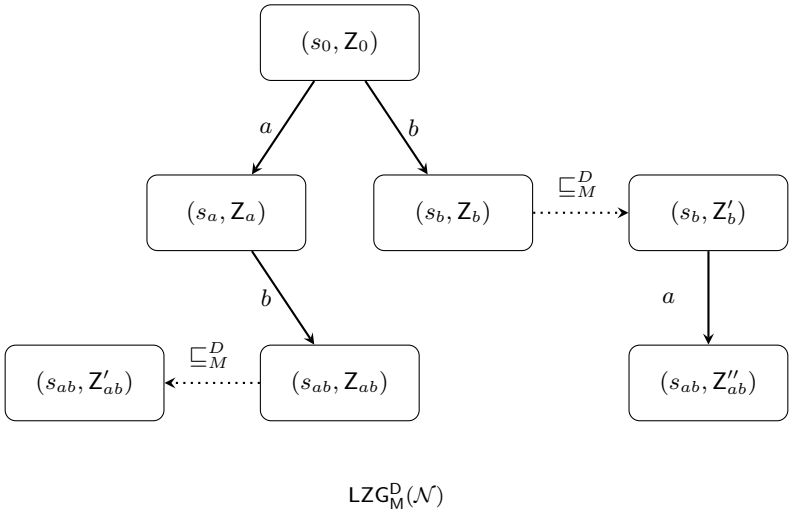


Figure 6.4: The diamond between a and b in LZG is torn apart by two subsumptions in $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}$

Let $\mathcal{N}$ be a network of timed automata and let a and b be actions of the network, such that $\mathsf{dom}(a) \cap \mathsf{dom}(b) = \emptyset$. Then, from Lemma 4.7, we know that there is a diamond in $\mathsf{LZG}(\mathcal{N})$ between the actions a and b. Consider the $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}(\mathcal{N})$ given in Figure 6.4. Suppose that the node (s_b, Z_b) is covered by (s_b, Z'_b). Further, suppose that a is an outgoing action from (s_b, Z'_b) and leads to a node $(s_{ab}, \mathsf{Z}''_{ab})$. Observe that the node $(s_{ab}, \mathsf{Z}_{ab})$ may be covered by another node $(s_{ab}, \mathsf{Z}'_{ab})$ in $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}(\mathcal{N})$. It is possible that the node $(s_{ab}, \mathsf{Z}'_{ab})$ may not be covered by the node $(s_{ab}, \mathsf{Z}''_{ab})$, or vice versa. Thus, in $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}(\mathcal{N})$, effectively, the diamond between a and b in $\mathsf{LZG}(\mathcal{N})$ is torn apart by two subsumptions, as illustrated in Figure 6.4.

Despite the lack of diamonds in $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}$, Theorem 6.1 allows us to apply any source set based method on $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}(\mathcal{N})$. Suppose that we have an algorithm that given a network $\mathcal{N}$ computes for every state q, the set of actions $\mathsf{source}(q)$. If this source function is complete for $\mathsf{LZG}(\mathcal{N})$ then we can use it for exploration in $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}(\mathcal{N})$. By Theorem 6.1, there is a source path in $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}(\mathcal{N})$ executing the given action α iff there is a source path in $\mathsf{LZG}(\mathcal{N})$. By Theorem 4.1 the later statement is equivalent to $\mathcal{N}$ having a run executing α.

6.4. Testing $\mathsf{Z} \not\subseteq \mathfrak{a}^*_M(\mathsf{Z}')$

The results of Section 6.3 imply that we can use $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}(\mathcal{N})$ for checking reachability provided $\mathcal{N}$ is D-spread-bounded and conforms to the bounds function M. In this section, we address the question of constructing $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}(\mathcal{N})$ efficiently. In principle, this can be done with a standard exploration algorithm if the subsumption relation $\mathfrak{a}^D_M$ can be computed efficiently. We show that checking $\mathfrak{a}^D_M$ subsumption between two local zones can be done with efficiency similar to that of testing subsumption in the standard setting.

In this section, we show that given two local zones Z and Z', checking if $\mathsf{Z} \sqsubseteq^D_M \mathsf{Z}'$ can be done in $\mathcal{O}(|\widetilde{X}'|^2)$ time, where $\widetilde{X}'$ is the set of all clocks (including offset clocks and reference clocks). Observe that the complexity is the same as in the global case, although in our setting we need to take into account the reference clocks. As in the global case, the complexity follows by showing that if inclusion does not hold, then there is a negative cycle witness over just two clocks, as shown in Proposition 6.2. However, the final test (Theorem 6.2) has subtle differences from the previous test on global zones.

Recall that in the representation of global zones using difference bound matrices (see Section 2.5), a special zero clock is added whose value is always 0. The value of a clock x in the global situation is read with respect to this zero clock as $x - 0$. In the local-time situation, the reference clocks act as different zeroes and hence there is a choice of which $\widetilde{x} - t_q$ to pick. In the inclusion test, we need to carefully choose this offset. The goal of this section is to provide the actual inclusion test through a sequence of intermediate technical observations, closely resembling the global test.

First, we state Lemma 6.14, that follows directly from the definition of the abstraction (see Definition 6.5).

Lemma 6.14. *Let* Z, Z' *be non-empty local zones. We have that* $\mathsf{Z} \not\subseteq \mathfrak{a}^*_M(\mathsf{Z}')$ *iff there exists a valuation* $\mathsf{v} \in \mathsf{Z}$ *such that* $\lceil \mathsf{v} \rceil^{M^*} \cap \mathsf{Z}'$ *is empty.*

Our objective is to efficiently find a valuation v that witnesses the non-inclusion. For the rest of this section, we consider zones to be represented using (canonical) distance graphs (see Section 2.5). Recall that in a distance graph, an edge $x \xrightarrow{(\prec, c)} y$ translates to $y - x \prec c$. Note that we use $\lceil x \rceil$ to denote the smallest integer greater than or equal to x.

Definition 6.11 (Local zone representation for a region)**.** Given a local valuation v, define local zone G_{v} to be the canonical form of the following set of constraints:

1. for each process A_p and clock $x \in X_p$ such that $x \notin \mathsf{Bounded}(\mathsf{v})$, add constraint $t_p - \widetilde{x} > M_x$ if $M_x \neq -\infty$ (which translates to edge

$t_p \xrightarrow{(<,-M_x)} \widetilde{x}$) or the constraint $t_p - \widetilde{x} \geq 0$ if $M_x = -\infty$ (which translates to edge $t_p \xrightarrow{(\leq,0)} \widetilde{x}$).

2. for every $x, y \in \mathsf{Bounded}(\mathsf{v})$, add $x - y < \lceil \mathsf{v}(x-y) \rceil$ if $\mathsf{v}(x-y)$ is not an integer, and $x - y \leq \mathsf{v}(x-y)$ otherwise (which gives edge $y \to x$ with the appropriate weight)

Lemma 6.15. *For each valuation* v*, we have* $[\![G_\mathsf{v}]\!] = [\mathsf{v}]^{M^*}$.

Proof. Follows directly from definitions. □

We now define a ceiling function for weights.

Definition 6.12. Given a weight $(\leq, c)$ with $c \in \mathbb{R}_{\geq 0}$, we define $\lceil (\leq, c) \rceil = (<, \lceil c \rceil)$ if c is not an integer, and $\lceil (\leq, c) \rceil = (\leq, c)$ otherwise. When the weight is of the form $(<, c)$ we define $\lceil (<, c) \rceil = (<, \lceil c \rceil)$ if c is not an integer, and $\lceil (<, c) \rceil = (<, c+1)$ otherwise.

We remark that there is some asymmetry in Definition 6.12 when the weight is of the form $(<, c)$. The reason for adopting this convention will be clear from the proof of Lemma 6.17, where we use it for the first time. A more detailed discussion about the choice of this convention is available in in [Sri12].

Lemma 6.16. *In the canonical distance graph* G_v*, the edge* $y \to x$ *(the tightest constraint for* $x - y$ *in* $[\mathsf{v}]^{M^*}$*) is given by:*

$$[\mathsf{v}]^{M^*}_{yx} = \begin{cases} (<, \infty) & y \notin \mathsf{Bounded}(\mathsf{v}) \\ \lceil (\leq, \mathsf{v}(t_p - y)) \rceil + (<, -M_x) & y \in \mathsf{Bounded}(\mathsf{v}),\ x \notin \mathsf{Bounded}(\mathsf{v}),\ M_x \neq -\infty \\ \lceil (\leq, \mathsf{v}(t_p - y)) \rceil & y \in \mathsf{Bounded}(\mathsf{v}),\ x \notin \mathsf{Bounded}(\mathsf{v}),\ M_x = -\infty \\ \lceil (\leq, \mathsf{v}(x - y)) \rceil & \textit{otherwise} \end{cases}$$

where t_p *is the reference clock of the process* A_p *such that* $x \in X_p$.

Proof. Let the graph obtained from constraints given in Definition 6.11 be called G'_v. This graph on canonicalization gives G_v.

The lemma follows from a sequence of observations on G'_v. If y is unbounded then there there is no outgoing edge from y so there is no path to x. So suppose that y is bounded. If x is also bounded then actually the shortest path from y to x in G'_v is given by the edge $y \to x$. To see this, assume to the contrary, that there is a shorter path $y \to \cdots \to x$. Together with the edge $x \to y$ this path would form a negative cycle.

The last case is when x is unbounded. Then, the only edge containing x is coming from its reference clock: it is $t_p \xrightarrow{(<,-M_x)} x$ or $t_p \xrightarrow{(\leq,0)} x$. So, all the paths from y to x use the edge $t_p \to x$. Since t_p is bounded by definition, by the first observation, the shortest path from y to t_p has weight $\lceil (\leq, \mathsf{v}(t_p - y)) \rceil$. □

Proposition 6.2. *Let* Z' *be an arbitrary local zone. Then,* $[\mathsf{v}]^{M^*} \cap \mathsf{Z}'$ *is empty iff there exist two variables* x, y *such that:*

1. $y \in \mathsf{Bounded}(\mathsf{v})$*,* x *is either a reference clock or a process clock with* $M_x \neq -\infty$ *and*
2. $[\mathsf{v}]^{M^*}_{yx} + \mathsf{Z}'_{xy} < (\leq, 0)$.

Proof. We assume that zones $[\mathsf{v}]^{M^*}$ and Z' are represented by their canonical distance graphs G_v and $G_{\mathsf{Z}'}$. The intersection $[\mathsf{v}]^{M^*} \cap \mathsf{Z}'$ is empty iff there is a negative cycle in $\min(G_\mathsf{v}, G_{\mathsf{Z}'})$. Let us call this negative cycle N. Since both the zones are canonical, we can assume that the edges of N alternate between those of G_v and $G_{\mathsf{Z}'}$. We will now establish that this cycle can be reduced to a cycle containing only two variables.

Consider an unbounded variable $x \in N$ with $M_x \neq -\infty$. The only edges in G_v involving x are of the form $y \to x$ with $y \in \mathsf{Bounded}(\mathsf{v})$. Since the consecutive edges of N should alternate between G_v and G_Z, this implies that the incoming edge should be from a bounded variable and the outgoing edge should be to a bounded variable. Thus, we cannot have two consecutive unbounded variables in N.

Suppose that there are two variables $b_1, b_2 \in N$ with $b_1, b_2 \in \mathsf{Bounded}(\mathsf{v})$. First, note that the constraints between b_1, b_2 in G_v can take two forms: either

1. $b_1 \xrightarrow{(\leq, c)} b_2$ and $b_2 \xrightarrow{(\leq, -c)} b_1$, or
2. $b_1 \xrightarrow{(<, c)} b_2$ and $b_2 \xrightarrow{(<, -c+1)} b_1$.

Consider the path $b_1 \to x_1 \cdots x_k \to b_2$ in N. Let the weight of this path be $(\prec, d)$. If $(\prec, d)$ is bigger than or equal to weight of the edge $b_1 \to b_2$ of G_v, then this sequence can be replaced with the edge $b_1 \to b_2$ to get a shorter negative cycle. If not, $(\prec, d)$ is strictly less than the weight of $b_1 \to b_2$: that is, $(\prec, d) < (\leq, c)$ or $(\prec, d) < (<, c)$ depending on case 1 or case 2 above. This entails that $(\prec, d)$ plus the weight of the edge $b_2 \to b_1$ from G_v is a negative cycle. Hence $b_1 \to x_1 \cdots x_k \to b_2 \to b_1$ with $b_2 \to b_1$ from G_v is a shorter negative cycle. Due to this reason, we can assume, without loss of generality, that N contains at most two variables from $\mathsf{Bounded}(\mathsf{v})$, and when there are exactly two of them they are connected by a direct edge, implying that N contains only these two variables.

Thus, we now have a cycle $b_1 \to x \to b_2 \to b_1$ in N. Since the edges of N should alternate between G_v and G_Z, we know that two consecutive edges of this cycle should be from the same graph. Since both G_v and G_Z are canonical, we can replace these two edges with a direct edge, to get a cycle of the same weight. Thus, we now have a cycle over two vertices, where at least one of them is a bounded variable.

Finally, consider an unbounded variable $x \in N$ with $M_x = -\infty$. Again, the only edges in G_{v} involving x are of the form $y \to x$ with $y \in \mathsf{Bounded}(\mathsf{v})$. The weight of $y \to x$ equals the path $y \to t_p \xrightarrow{(\leq,0)} x$. Replacing the edge with this path still gives a negative cycle. Now, note that in zone Z', every valuation satisfies $x - t_p \leq 0$. Therefore the weight $\mathsf{Z}'_{t_p x} \leq (\leq, 0)$ and replacing $t_p \xrightarrow{(\leq,0)} x$ with the edge $t_p \xrightarrow{(\prec,\mathsf{Z}'_{t_p x})} x$ also gives a negative cycle. The original N had an outgoing edge $x \to x_1$ coming from Z'. Since Z' is canonical, the edges $t_p \to x \to x_1$ can be replaced with $t_p \to x_1$ with weight coming from Z'. This eliminates x from N.

These observations reduce the negative cycle to the form required by the lemma. □

We will now try to compute the least value of $[\mathsf{v}]^{M^*}_{yx}$ for $\mathsf{v} \in \mathsf{Z}$. Note that for $\mathsf{v} \in \mathsf{Z}$ such that $\mathsf{v}(t_q - y) > M_y$, the value of $[\mathsf{v}]^{M^*}_{yx}$ is $(<, \infty)$ by Lemma 6.16. So, for these valuations, the inequality 2 never holds. Therefore, we first look at valuations in $\mathsf{Z} \cap (t_q - y \leq M_y)$. We consider two cases, depending on whether x is bounded. We say that a valuation v is x-bounded if $x \in \mathsf{Bounded}(\mathsf{v})$; otherwise, we say that v is x-unbounded.

Lemma 6.17. *When $\mathsf{Z} \cap (t_q - y \leq M_y)$ is non-empty and has only x-bounded valuations, the least value of $[\mathsf{v}]^{M^*}_{yx}$ for $\mathsf{v} \in \mathsf{Z}$ is given by $\lceil -\mathsf{Z}_{xy} \rceil$.*

Proof. Let G_1 be the distance graph obtained from G_{Z} by replacing the weight of the edge $y \to t_q$ with $\min(\mathsf{Z}_{yt_q}, (\leq, M_y))$. The set $[\![G_1]\!]$ equals $\mathsf{Z} \cap (t_q - y \leq M_y)$ and contains only x-bounded valuations by assumption. From Lemma 6.16, we know that the value of $[\mathsf{v}]^{M^*}_{yx}$ for $\mathsf{v} \in [\![G_1]\!]$ is given by $\lceil (\leq, \mathsf{v}(x - y)) \rceil$. To find the least value of this quantity, we need the largest value of $\mathsf{v}(y - x)$ among valuations in $[\![G_1]\!]$. This value can be inferred from the weight $(\preccurlyeq_1, w_1)$ of the shortest path from x to y in G_1.

Here, the inequality $\preccurlyeq_1$ could be a weak inequality ($\preccurlyeq_1 = \leq$) or a strong inequality ($\preccurlyeq_1 = <$). When the inequality is weak, this means that for all $\mathsf{v} \in [\![G_1]\!]$, we have $\mathsf{v}(y - x) \leq w_1$. This implies that for all $\mathsf{v} \in [\![G_1]\!]$, we have $\mathsf{v}(x - y) \geq -w_1$. In this case, the least value of $\lceil (\leq, \mathsf{v}(x - y)) \rceil$ is given by $(\leq, -w_1)$, which is equal to $\lceil -\mathsf{Z}_{xy} \rceil$ according to Definition 6.12.

For the case of strong inequality, the situation is slightly more complicated. In this case, we know that for all $\mathsf{v} \in [\![G_1]\!]$, $\mathsf{v}(y - x) < w_1$. This implies that for all $\mathsf{v} \in [\![G_1]\!]$, we have $\mathsf{v}(x - y) > -w_1$. This means that for all $\mathsf{v} \in [\![G_1]\!]$, we have $\mathsf{v}(x - y) \geq -w_1 + \varepsilon$, for some $\varepsilon < 1$. In this case, the least value of $\lceil (\leq, \mathsf{v}(x - y)) \rceil$ is equal to $(\leq, \lceil -w_1 + \varepsilon \rceil)$, which is given by $(\leq, -w_1 + 1)$. This is again equal to $\lceil -\mathsf{Z}_{xy} \rceil$, as per Definition 6.12.

Observe that the shortest path from x to y in G_{Z} also has weight $(\preccurlyeq_1, w_1)$. The differences between G_1 and G_{Z} are only due to the modification of the weight of the edge $y \to t_p$. But a shortest weight path from x to y cannot

take the edge $y \to t_p$ as this would produce a path in which y appears twice. Hence $(\prec_1, w_1) = \mathsf{Z}_{xy}$ and the lemma follows. □

Lemma 6.18. *Assume that $M_x \neq -\infty$. When $\mathsf{Z} \cap (t_q - y \leq M_y)$ is non-empty and has only x-unbounded valuations, the least value of $[\mathsf{v}]^{M^*}_{yx}$ from $\mathsf{v} \in \mathsf{Z}$ is given by $(<, -M_x) + \lceil -\mathsf{Z}_{t_p y} \rceil$.*

Proof. The proof proceeds as in Lemma 6.17, with the change that now, by Lemma 6.16, the value of $[\mathsf{v}]^{M^*}_{yx}$ is given by $\lceil (\leq, \mathsf{v}(t_p - y)) \rceil + (<, -M_x)$. We need the least value of $\mathsf{v}(t_p - y)$ and hence the greatest value of $\mathsf{v}(y - t_p)$. As in the proof of Lemma 6.17, consider the graph G_1 obtained by replacing the weight of the edge $y \to t_q$ by $\min(\mathsf{Z}_{yt_q}, (\leq, M_y))$ in G_{Z}. The shortest path from t_p to y is given by $\mathsf{Z}_{t_p y}$ and hence the least value of $[\mathsf{v}]^{M^*}_{yx}$ is $\lceil -\mathsf{Z}_{t_p y} \rceil + (<, -M_x)$. □

Lemma 6.19. *Assume that $M_x \neq -\infty$. When $\mathsf{Z} \cap (t_q - y \leq M_y)$ contains both x-bounded and x-unbounded valuations, the least value of $[\mathsf{v}]^{M^*}_{yx}$ is given by:*

$$\begin{cases} \lceil -\mathsf{Z}_{xy} \rceil & \text{if } \mathsf{Z}_{xy} \leq (\leq, M_x) + \mathsf{Z}_{t_p y} \\ (<, -M_x) + \lceil -\mathsf{Z}_{t_p y} \rceil & \text{otherwise} \end{cases}$$

Proof. We will find the least value β_1 among x-bounded valuations and the least value β_2 among x-unbounded valuations and then take $\min(\beta_1, \beta_2)$.

To find β_1, consider the graph G'_1 obtained from G_{Z} by modifying the weight of edges $y \to t_q$ to $\min(\mathsf{Z}_{yt_q}, (\leq, M_y))$ and $x \to t_p$ to $\min(\mathsf{Z}_{xt_p}, (\leq, M_x))$. The set $[\![G'_1]\!]$ gives the set of x-bounded valuations in $\mathsf{Z} \cap (t_q - y \leq M_y)$. This set is non-empty by the assumption made in the lemma. Among these valuations, $[\mathsf{v}]^{M^*}_{yx}$ is given by $\lceil (\leq, \mathsf{v}(x - y)) \rceil$. We now proceed as in the proof of Lemma 6.17 to find the shortest path from x to y in G'_1. The newly added edge $x \to t_p$ could influence this, and we get the shortest path to be $\min(\mathsf{Z}_{xy}, (\leq, M_x) + \mathsf{Z}_{t_p y})$. Hence:

$$\beta_1 = \begin{cases} \lceil -\mathsf{Z}_{xy} \rceil & \text{if } \mathsf{Z}_{xy} \leq (\leq, M_x) + \mathsf{Z}_{t_p y} \\ (\leq, -M_x) + \lceil -\mathsf{Z}_{t_p y} \rceil & \text{otherwise} \end{cases}$$

To find β_2, consider the graph G'_2 obtained from G_{Z} by modifying edge $y \to t_q$ to $\min(\mathsf{Z}_{yt_q}, (\leq, M_y))$ and $t_p \to x$ to $\min(\mathsf{Z}_{t_p x}, (<, -M_x))$. The set $[\![G'_2]\!]$ gives the set of x-unbounded valuations in $\mathsf{Z} \cap (t_q - y \leq M_y)$. Again, this set is non-empty by the assumption made in the lemma. For these valuations, from Lemma 6.16, we know that $[\mathsf{v}]^{M^*}_{yx}$ is given by $\lceil \mathsf{v}(t_p - y) \rceil + (<, -M_x)$. We now proceed as in the proof of Lemma 6.18 to find the shortest path from t_p to y in G'_2. This is given by $\min(\mathsf{Z}_{t_p y}, (<, -M_x) + \mathsf{Z}_{xy})$. Observe that $\lceil (<, M_x) - \mathsf{Z}_{xy} \rceil = (<, M_x) - \mathsf{Z}_{xy} + (<, 1)$. Hence:

$$\beta_2 = \begin{cases} -\mathsf{Z}_{xy} + (<, 1) & \text{if } \mathsf{Z}_{t_p y} \geq (<, -M_x) + \mathsf{Z}_{xy} \\ \lceil -\mathsf{Z}_{t_p y} \rceil + (<, -M_x) & \text{otherwise} \end{cases}$$

We want to get β_1 and β_2 in terms of the same cases so that we can find the minimum.

When $\mathsf{Z}_{xy} \leq (\leq, M_x) + \mathsf{Z}_{t_py}$, we get $\mathsf{Z}_{xy} + (<, -M_x) \leq (<, 0) + \mathsf{Z}_{t_py}$ by adding $(<, -M_x)$ on both sides. This implies that $\mathsf{Z}_{xy} + (<, -M_x) \leq \mathsf{Z}_{t_py}$. Hence in this case $\min(\beta_1, \beta_2)$ is given by $\lceil -\mathsf{Z}_{xy} \rceil$.

When $\mathsf{Z}_{xy} > (\leq, M_x) + \mathsf{Z}_{t_py}$, there is a corner situation when still $\mathsf{Z}_{t_py} \geq (<, -M_x) + \mathsf{Z}_{xy}$ and otherwise $\mathsf{Z}_{t_py} < (<, M_x) + \mathsf{Z}_{t_py}$. In the latter case, $\min(\beta_1, \beta_2)$ is given by $\lceil -\mathsf{Z}_{t_py} \rceil + (<, -M_x)$. It remains to resolve the former case. This happens when $\mathsf{Z}_{xy} = (<, c)$, $\mathsf{Z}_{t_py} = (\leq, c - M_x)$. So β_1 is $(\leq, -M_x) + \lceil -\mathsf{Z}_{t_py} \rceil = (\leq, c)$ and β_2 is $-\mathsf{Z}_{xy} + (<, 1) = (<, c + 1)$. Hence $\min(\beta_1, \beta_2)$ is given by β_1. This proves the lemma. □

Proposition 6.3. *Assume that $M_x \neq -\infty$. The least value of $[\mathsf{v}]^{M^*}_{yx}$ among valuations in Z is given by:*

$$\begin{cases} (<, \infty) & \text{if } \mathsf{Z}_{t_qy} < (\leq, -M_y) \\ \max(\lceil -\mathsf{Z}_{xy} \rceil, (<, -M_x) + \lceil -\mathsf{Z}_{t_py} \rceil) & \text{otherwise} \end{cases}$$

Proof. When $\mathsf{Z}_{t_qy} < (\leq, -M_y)$, every valuation in Z is y-unbounded. From Lemma 6.16, the value of $[\mathsf{v}]^{M^*}_{yx}$ is $(<, \infty)$. This gives the first case in the lemma. Otherwise, Lemmas 6.17, 6.18 and 6.19 give this value based on whether $\mathsf{Z} \cap (t_q - y \leq M_y)$ contains only x-bounded, only x-unbounded or both kinds of valuations, respectively. In each of the cases, we can show that the value is given by the maximum of the two quantities required by the lemma. Let G' be the distance graph obtained by taking G_{Z} and modifying the weight of the $y \to t_q$ edge to $\min(\mathsf{Z}_{yt_q}, (\leq, M_y))$. The set of valuations $[\![G']\!]$ is $\mathsf{Z} \cap (t_q - y \leq M_y)$, which is non-empty in our case.

Suppose that $[\![G']\!]$ contains only x-bounded valuations, Lemma 6.17 gives the least value of $[\mathsf{v}]^{M^*}_{yx}$ to be $\lceil -\mathsf{Z}_{xy} \rceil$. Since every valuation in $[\![G']\!]$ is x-bounded, G' entails $t_p - x \leq M_x$ and the shortest path from x to t_p is at most $(\leq, M_x)$ - let us call this path G'_{xt_p}. We then have $G'_{xt_p} \leq (\leq, M_x)$. Now, consider the shortest path from x to y in G'. Call it G'_{xy}. By property of distance graphs, we have $G'_{xy} \leq G'_{xt_p} + G'_{t_py}$. The paths leading to y do not get affected by the addition of $y \to t_q$ to G_{Z}. Hence $G'_{xy} = \mathsf{Z}_{xy}$ and $G'_{t_py} = \mathsf{Z}_{t_py}$. This gives $\mathsf{Z}_{xy} \leq (\leq, M_x) + \mathsf{Z}_{t_py}$, which implies $\mathsf{Z}_{xy} < (<, M_x) + \mathsf{Z}_{t_py}$. Hence, $\lceil -\mathsf{Z}_{xy} \rceil = \max(\lceil -\mathsf{Z}_{xy} \rceil, (<, -M_x) + \lceil -\mathsf{Z}_{t_py} \rceil)$.

When $[\![G']\!]$ contains only x-unbounded valuations, Lemma 6.18 gives the least value of $[\mathsf{v}]^{M^*}_{yx}$ to be $(<, -M_x) + \lceil -\mathsf{Z}_{t_py} \rceil$. Since every valuation in $[\![G']\!]$ in x-unbounded, we have the shortest path $G'_{t_px} \leq (<, -M_x)$. Note that $G'_{t_py} \leq G'_{t_px} + G'_{xy}$. From the same argument as in the case for only x-bounded valuations, we have that $\mathsf{Z}_{t_py} \leq (<, -M_x) + \mathsf{Z}_{xy}$. Adding $(<, M_x)$ to both sides, we get $(<, M_x) + \mathsf{Z}_{t_py} \leq \mathsf{Z}_{xy}$, which implies that $(<, -M_x) - \mathsf{Z}_{t_py} \geq -\mathsf{Z}_{xy}$. Hence, $(<, -M_x) + \lceil -\mathsf{Z}_{t_py} \rceil = \max(\lceil -\mathsf{Z}_{xy} \rceil, (<, -M_x) + \lceil -\mathsf{Z}_{t_py} \rceil)$.

When $[\![G']\!]$ contains both x-bounded and x-unbounded valuations, Lemma 6.19 gives two cases. When $\mathsf{Z}_{xy} \leq (\leq, M_x) + \mathsf{Z}_{t_py}$, the least value of $[\mathsf{v}]^{M^*}_{yx}$

is $\lceil -\mathsf{Z}_{xy} \rceil$. This is equal to $\max(\lceil -\mathsf{Z}_{xy} \rceil, (<, -M_x) + \lceil -\mathsf{Z}_{t_p y} \rceil)$ by the same argument as in the only x-bounded case. When $\mathsf{Z}_{xy} > (\leq, M_x) + \mathsf{Z}_{t_p y}$, the least value of $[\mathsf{v}]^{M^*}_{yx}$ is $(<, -M_x) + \lceil -\mathsf{Z}_{t_p y} \rceil$. Now, since $\mathsf{Z}_{xy} > (\leq, M_x) + \mathsf{Z}_{t_p y}$, it is clear that $\mathsf{Z}_{xy} > (<, M_x) + \mathsf{Z}_{t_p y}$, where we have changed $(\leq, M_x)$ to $(<, M_x)$. We then have that $(<, -M_x) + \lceil -\mathsf{Z}_{t_p y} \rceil = \max(\lceil -\mathsf{Z}_{xy} \rceil, (< , -M_x) + \lceil -\mathsf{Z}_{t_p y} \rceil)$. □

Theorem 6.2. *Let* Z, Z' *be non-empty zones. Then,* $\mathsf{Z} \not\subseteq \mathfrak{a}^*_M(\mathsf{Z}')$ *iff there exist two variables* x, y *such that*

$$\mathsf{Z}_{t_q y} \geq (\leq, -M_y) \text{ and } \mathsf{Z}'_{xy} < \mathsf{Z}_{xy} \text{ and } \mathsf{Z}'_{xy} + (<, -M_x) < \mathsf{Z}_{t_p y}$$

where t_p, t_q *are reference clocks and* $x \in X_p \cup \{t_p\}$ *and* $y \in X_q \cup \{t_q\}$*, respectively.*

Proof. ($\Rightarrow$): Suppose that $\mathsf{Z} \not\subseteq \mathfrak{a}^*_M(\mathsf{Z}')$. Then, there exists $\mathsf{v} \in \mathsf{Z}$ such that $[\mathsf{v}]^{M^*} \cap \mathsf{Z}' = \emptyset$. By Proposition 6.2, this means there are two variables x, y with $y \in \mathsf{Bounded}(\mathsf{v})$ and $M_x \neq -\infty$ such that $[\mathsf{v}]^{M^*}_{yx} + \mathsf{Z}'_{xy} < (\leq, 0)$. Hence the valuation in Z with the least value of $[\mathsf{v}]^{M^*}_{yx}$ satisfies this constraint. Proposition 6.3 gives this least value. Plugging this value to this constraint and noting that $\lceil (\preccurlyeq, -d) \rceil + (\preccurlyeq', d') < (\leq, 0)$ iff $(\preccurlyeq', d') < (\preccurlyeq, d)$ gives the right hand side of the theorem.

($\Leftarrow$): When the three conditions in the right hand side of the theorem hold, then the valuation v with the least value of $[\mathsf{v}]^{M^*}_{yx}$ as given by Proposition 6.3 satisfies $[\mathsf{v}]^{M^*}_{yx} + \mathsf{Z}'_{xy} < (\leq, 0)$. This implies that $[\mathsf{v}]^{M^*} \cap \mathsf{Z}' = \emptyset$ and hence $\mathsf{Z} \not\subseteq \mathfrak{a}^*_M(\mathsf{Z}')$. □

Observe that the variables x, y in the check in Theorem 6.2 can denote the reference clocks t_p, t_q also respectively. In that case, $M_x = \infty, M_y = \infty$ respectively. We now propose a simplified version of the test in Theorem 6.2 that deals with the cases where x and y are reference clocks separately.

Corollary 6.2. *Let* Z, Z' *be non-empty zones. Then,* $\mathsf{Z} \not\subseteq \mathfrak{a}^*_M(\mathsf{Z}')$ *iff there exist two variables* $x \in X_p \cup \{t_p\}, y \in X_q \cup \{t_q\}$ *such that*

$$\begin{array}{ll}
\mathsf{Z}_{t_q y} \geq (\leq, -M_y) \text{ and } \mathsf{Z}'_{xy} < \mathsf{Z}_{xy} \text{ and } \mathsf{Z}'_{xy} + (<, -M_x) < \mathsf{Z}_{t_p y} & \text{if } x \neq t_p, y \neq t_q \\
\mathsf{Z}_{t_q y} \geq (\leq, -M_y) \text{ and } \mathsf{Z}'_{t_p y} < \mathsf{Z}_{t_p y} & \text{if } x = t_p, y \neq t_q \\
\mathsf{Z}'_{x t_q} < \mathsf{Z}_{x t_q} \text{ and } \mathsf{Z}'_{x t_q} + (<, -M_x) < \mathsf{Z}_{t_p t_q} & \text{if } x \neq t_p, y = t_q \\
\mathsf{Z}'_{t_p t_q} < \mathsf{Z}_{t_p t_q} & \text{if } x = t_p, y = t_q
\end{array}$$

Chapter 7

Spread-bounded systems

In Chapter 6, we proposed a subsumption relation $\sqsubseteq_M^D$ for local zones that is based on a simulation relation between local valuations, and introduced the transition system obtained by applying the $\sqsubseteq_M^D$ subsumption to the local zone graph of a network, referred to as the $\mathsf{LZG}_\mathsf{M}^\mathsf{D}$ of the network. We showed that if a network of timed automata is D-spread-bounded, then it is sufficient to explore the $\mathsf{LZG}_\mathsf{M}^\mathsf{D}$ of the network to answer the reachability problem for the network. Furthermore, using Theorem 6.1, we showed that the properties of $\mathsf{LZG}_\mathsf{M}^\mathsf{D}$ allow us to use a partial order reduction technique while exploring the $\mathsf{LZG}_\mathsf{M}^\mathsf{D}$ of the network. In Chapter 8, we will propose an algorithm that applies a partial order reduction method based on source sets to the $\mathsf{LZG}_\mathsf{M}^\mathsf{D}$ of a network.

However, constructing the $\mathsf{LZG}_\mathsf{M}^\mathsf{D}$ of a network of timed automata (and then applying partial order reduction to the $\mathsf{LZG}_\mathsf{M}^\mathsf{D}$) is sound and complete with respect to reachability only if the network is D-spread-bounded. Therefore, it is crucial to have ways to identify when a network is spread bounded and compute a tight bound D on the spread of the network.

In this chapter, we first illustrate, by the means of an example, that there are networks of timed automata which have unbounded spread. This demonstrates the need for efficiently checkable conditions to say when a network is spread-bounded. We define two classes of networks of timed automata, which we refer to as *global-local systems* and *client-server systems*, respectively. We give customized conditions for spread-boundedness of networks belonging to both these classes.

A major handicap of the $\sqsubseteq_M^D$ subsumption is that in general, it preserves more information than the sync-subsumption. For subsumption between two nodes of the LZG, while the sync-subsumption only compares synchronized valuations in the zones, the $\sqsubseteq_M^D$ subsumption involves comparing all the valuations of some fixed spread D. As a consequence, the $\sqsubseteq_M^D$ subsumption may result in lesser coverings and consequently, a bigger zone graph, when compared to the zone graph generated using the sync-subsumption. Thus,

if a network of timed automata is 0-spread-bounded, we have the best of both worlds - we are able to work with smaller zone graphs in the first place, while also benefiting from the gains due to partial order reduction.

Lastly, we show that, in general, the problem of deciding whether a network of timed automata is 0-spread-bounded is hard. We prove that given a network of timed automata, deciding whether the network is 0-spread-bounded is PSPACE-complete. We also show that deciding whether a network of timed automata is D-spread-bounded is PSPACE-hard.

Definitions: Here we briefly recall some of the definitions pertaining to this chapter. Let $\mathcal{N} = \{A_1, A_2, \cdots, A_n\}$ be a network of timed automata. The spread between processes A_p and A_q of $\mathcal{N}$ in a valuation v is defined as the absolute value of the difference between their reference clocks, $|\mathsf{v}(t_p) - \mathsf{v}(t_q)|$. We say that a valuation v has spread D if the spread between every pair of processes in v is at most D. Given a local run σ of $\mathcal{N}$, we say that σ is *D-spread* if all valuations of σ have spread D. We say that $\mathcal{N}$ is said to be *D-spread-bounded* if every local run of $\mathcal{N}$ can be converted to a D-spread run by adjusting the delays.

7.1. Systems with unbounded spread

In this section, we present an example of a network of timed automata that has unbounded spread. In doing so, we illustrate a typical situation that prevents a system from having a bounded spread.

Consider a network $\mathcal{A}$ that consists of two processes, A_1 and A_2, as shown in Figure 7.1.

Lemma 7.1. *$\mathcal{A}$ is not spread-bounded.*

Proof. Consider the run $a^m bc$ of $\mathcal{A}$. In this run, the process A_1 executes m times the self-loop a on state p_0. Note that each iteration of this loop takes exactly 1 time unit. In parallel, the process A_2 needs to execute a local action b in a bounded time (without elapsing more than 1 time unit) and then synchronize with A_1. So, A_2 reaches q_1 in less than 1 time unit. As a consequence, after executing the loop on action a for m times, the spread between A_1 and A_2 is at least $m - 1$. The two automata are in states p_0 and q_0, and the local time of A_1 is at least m and the local time of A_2 is less than 1. Thus, the run $a^m bc$ is feasible in local time semantics and has spread at least $m - 1$. Since m can be arbitrary, the spread of this network is not bounded. □

This example shows that not all networks of timed automata are spread-bounded. Thus, if we are to apply partial order reduction method to a

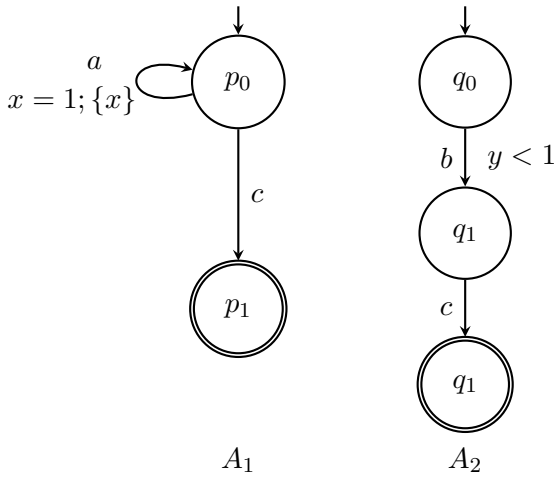


Figure 7.1: Example of a network of timed automata with unbounded spread

network of timed automata, we need to be sure that the network is spread-bounded. Therefore, it is crucial to have ways to identify when a network is spread-bounded and compute the bound D on the spread of the network. This is our focus in the next sections.

We define two classes of networks of timed automata, namely *global-local systems* and *client-server systems*. We then propose conditions for spread-boundedness for networks belonging to these classes and give bounds for the spread of such networks.

7.2. Global-local systems

We introduce a special class of networks of timed automata where each communication action synchronizes all the components of the network. We say a network $\mathcal{N} = \{A_1, A_2, \cdots, A_n\}$ is a *global-local system* if it satisfies two properties:

(gl-trans) $\mathcal{N}$ has only two kinds of actions - *local actions* whose domain is $\{A_i\}$ for some i, and *global actions* whose domain is $\{A_1, A_2, \cdots, A_n\}$.

(gl-final) Every accepting state of $\mathcal{N}$ is only reachable by executing a global action in the end. In other words, if a tuple $(q_1, \cdots, q_n)$ is an accepting state, then for $1 \leq i \leq n$ each incoming transition to q_i is labelled by a global action.

An execution of a global-local system is a sequence that alternates between subsequences containing only local actions and subsequences containing only

global actions. From Lemma 4.7, we know that in a sequence of local actions, actions of different processes can be commuted as they have disjoint domains. However, no local action can be commuted with a global action, as they do not have disjoint domains. We also cannot commute two global actions. Effectively, this means that a global action cannot be commuted with any other action. As a consequence, any commuting of actions that we do has to be restricted to the sequence of local actions between two global actions.

We will now give sufficient conditions for a global-local system to be spread-bounded. In Section 7.2.1, we focus on conditions for 0-spread-boundedness. Later, in Section 7.2.2 we will discuss conditions to conclude that a global-local system is spread-bounded, but not necessarily 0-spread-bounded.

7.2.1 Conditions for 0-spread

In this section, we investigate when a global-local system is 0-spread. We propose some sufficient conditions for a global-local system to be 0-spread.

By definition, the valuation obtained immediately after executing a global action has spread 0. We observe that one way to make a run 0-spread is to make the execution times of all local actions between two global actions identical. A local fragment of the run can be converted to this form, if one of the following are possible:

1. all local actions can be delayed so that they can be executed at the time of the next global action, or

2. all local actions can be accelerated so that they can be executed at the time of the preceding global action.

If it is possible to transform every sequence of local actions in this way, then, the whole run can be converted to a 0-spread run.

As a preparatory step, for every local state of a process and every clock, we calculate if there is an active, upper or lower bound guard in this state. This is analogous to computing LU bounds for clocks with the exception that here, we only need to know if the bound is $-\infty$ or not; we do not care for the exact value of the bound.

Definition 7.1. For a local state q of a process, we define $UGV(q)$ to be the set of clocks x such that there is a path from q to a transition with an upper bound guard on x, and x is not reset on the path. Similarly we define $LGV(q)$ but for lower bound guards.

Definition 7.2. A state q of an automaton is *locally-relaxed* if for each sequence of local actions $q = q_1 \xrightarrow[R_1]{g_1} q_2 \xrightarrow[R_2]{g_2} \dots \xrightarrow[R_k]{g_k} q_{k+1}$:

1. If g_i has an upper bound guard on x, then $x \in R_1 \cup \dots \cup R_{i-1}$; in particular, g_1 has no upper bound guard.

2. If $x \in R_i$ then x does not have a lower bound guard in $g_{i+1}, \ldots, g_k$, and $x \notin LGV(q_{k+1})$.

A state is *locally-tense* if

1. there are no lower bound guards in $g_1, \ldots, g_k$, and
2. If $x \in R_1 \cup \cdots \cup R_k$ then $x \notin UGV(q_{k+1})$.

Our strategy is to convert a run to a 0-spread run without changing the execution times of global actions. For locally-relaxed states, we will execute all the local actions just before the next global action. Condition 1 says that there should be no upper bound guard that could prevent executing a transition later than in the original run. Condition 2 ensures that there are no problems later; since we reset x later than in the original run, in the future we do not allow lower bounds. For example, consider the following run $\xrightarrow[\{x\}]{} \xrightarrow{x>10}$. Observe that this run requires an elapse of 10 units of time between the first and the second transitions - as a consequence, the two transitions cannot both be executed just before the global action. Observe that the condition that we discussed disqualifies such a sequence.

For locally-tense states the idea is to execute all local actions immediately after the preceding global action. So, we need condition 1 that guarantees that there are no lower bound guards in the sequence, to be sure that we are allowed to do this. Since we execute local actions sooner, the resets happen sooner. This should not cause any problem for the rest of the run. Condition 2 ensures this by eliminating the possibility of upper bound guards involving a clock that is reset in the rest of the run.

Having defined the notion of locally-relaxed and locally-tense states, we now introduce the notion of a *tame* global-local system, in which all reachable states are of one of these two forms.

Definition 7.3. A global-local system is *tame* iff every reachable configuration consists entirely of locally-relaxed states or entirely of locally-tense states.

We will now show that if a global-local system is tame, then it is 0-spread.

Theorem 7.1. *If $\mathcal{A}$ is a tame global-local system, then $\mathcal{A}$ is 0-spread.*

Proof. Consider a run of $\mathcal{A}$:

$$(q_0, \mathsf{v}_0) \xrightarrow{u_1}{}^* (q_1, \mathsf{v}_1) \Longrightarrow^{w_1} \cdots (q_{2n-2}, \mathsf{v}_{2n-2}) \xrightarrow{u_n}{}^* (q_{2n-1}, \mathsf{v}_{2n-1}) \Longrightarrow^{w_n} (q_{2n}, \mathsf{v}_{2n})$$

Here u_i's are sequences of local actions and w_i's are sequences of global actions (denoted by $\Longrightarrow$ arrows). Observe that the valuation obtained after the execution of each action in a sequence w_i of global actions has spread 0 by definition. Therefore, we do not need to modify the time of execution of

any action in w_i's. We need to convert the sequences u_i of local actions to a 0-spread run. We will execute each action in a sequence u_i at the same time. If the configuration reached after w_{i-1} is locally-tense, we will execute each action in u_i at the time of execution of the last action in w_{i-1}. Otherwise, we will execute each action in u_i at the time of execution of the first action of w_i.

Consider one such sequence of local actions $(q_i, \mathsf{v}_i) \xrightarrow{u_{i+1}}{}^{*} (q_{i+1}, \mathsf{v}_{i+1})$. We assume that we have managed to construct a 0-spread run till (q_i, v'_i) and maintain following invariant.

Invariant 7.1.1. (q_i, v'_i) satisfies the following

1. $\mathsf{v}'_i(t_p) = \mathsf{v}_i(t_p)$ for each reference clock t_p,
2. if there are active $x > c$ bounds from q_i, then $\mathsf{v}'_i(\widetilde{x}) \leq \mathsf{v}_i(\widetilde{x})$,
3. if there are active $x < c$ bounds from q_i, then $\mathsf{v}'_i(\widetilde{x}) \geq \mathsf{v}_i(\widetilde{x})$.

Note that here, by active, we mean active in the graph of one of the processes. The local run u_{i+1} looks as follows:

$$(q_{i,0}, \mathsf{v}_{i,0}) \xrightarrow{\delta_1} \xrightarrow[R_1]{g_1} (q_{i,1}, \mathsf{v}_{i,1}) \xrightarrow{\delta_2} \xrightarrow[R_2]{g_2} \dots \xrightarrow{\delta_l} \xrightarrow[R_l]{g_l} (q_{i,l}, \mathsf{v}_{i,l})$$

where $q_{i,0} = q_i$, $\mathsf{v}_{i,0} = \mathsf{v}_i$ and $q_{i,l} = q_{i+1}$, $\mathsf{v}_{i,l} = \mathsf{v}_{i+1}$.

Suppose q_i consists of locally-tense states. We replace the run u_{i+1} with the following run:

$$(q_{i,0}, \mathsf{v}'_{i,0}) \xrightarrow{0} \xrightarrow[R_1]{g_1} (q_{i,1}, \mathsf{v}'_{i,1}) \xrightarrow{0} \xrightarrow[R_2]{g_2} \dots \xrightarrow{0} \xrightarrow[R_l]{g_l} (q_{i,l}, \mathsf{v}'_{i,l}) \xrightarrow{\delta} (q_{i,l}, \mathsf{v}'_{i+1})$$

where $\mathsf{v}'_{i,0} = \mathsf{v}'_i$. The difference is that we do not have any delay between consecutive actions, and just do an accumulated delay at the end.

We will show the following claim.

Invariant 7.1.2. All pairs $\mathsf{v}_{i,j}$, $\mathsf{v}'_{i,j}$ satisfy the following

1. $\mathsf{v}'_{i,j}(t_p) \leq \mathsf{v}_{i,j}(t_p)$ for each reference clock t_p,
2. if there is an active $x < c$ bound from q_i, then $\mathsf{v}'_{i,j}(\widetilde{x}) \geq \mathsf{v}_{i,j}(\widetilde{x})$, or $x \in R_1, \dots, R_{j-1}$ and $\mathsf{v}'_{i,j}(\widetilde{x}) = \mathsf{v}'_{i,j}(t_p)$, where x is a clock of process A_p.

Observe that this invariant allows us to execute the sequence, since by our assumption, all guards in $g_1, \dots, g_l$ are upper bound guards.

The invariant clearly holds for $j = 0$, as we have assumed that the pair v_i, v'_i satisfies a stronger assumption 7.1.1. For $j \geq 1$ we have the first item immediately from the definition. The second item holds by easy analysis.

In this way, we arrive at the pair $\mathsf{v}_{i,l}$, $\mathsf{v}'_{i,l}$ satisfying the above two conditions. We consider δ such that $\mathsf{v}'_{i+1}(t_p) = \mathsf{v}_{i+1}(t_p)$, for all reference clocks t_p. This establishes the first item from the general invariant.

For the second item, if $x \geq c$ is active in q_{i+1}, we have two cases. If $x \in R_1, \ldots, R_l$, then $\mathsf{v}'_{i+1}(\widetilde{x}) = \mathsf{v}'_i(t_p) = \mathsf{v}_i(t_p)$ and $\mathsf{v}_{i+1}(\widetilde{x}) \geq \mathsf{v}_i(t_p)$. As a consequence, we have $\mathsf{v}'_{i+1}(\widetilde{x}) \leq \mathsf{v}_{i+1}(\widetilde{x})$. If x is not reset, then x had an active upper bound already in q_i, so we can use the induction assumption.

For the third item, if $x \leq c$ is active in q_{i+1}, then $x \in UGV(q_{i+1})$. As a consequence, x is not reset in $R_1, \ldots, R_l$ and so, we use the induction assumption. Thus, we are done with the case when q_i consists of locally-tense states.

Next, we consider the case where q_i consists entirely of locally-relaxed states. We replace the original run u_{i+1} with the following run:

$$(q_{i,0}, \mathsf{v}'_i) \xrightarrow{\delta} (q_{i,0}, \mathsf{v}'_{i,0}) \xrightarrow{0} \xrightarrow[R_1]{g_1} (q_{i,1}, \mathsf{v}'_{i,1}) \xrightarrow{0} \xrightarrow[R_2]{g_2} \ldots \xrightarrow{0} \xrightarrow[R_l]{g_l} (q_{i,l}, \mathsf{v}'_{i,l})$$

where $\mathsf{v}'_{i,l} = \mathsf{v}'_{i+1}$. Observe that we first do the accumulated delay between the two global actions and then do all local actions in 0-time. We will show the following.

Invariant 7.1.3. All pairs $\mathsf{v}_{i,j}$, $\mathsf{v}'_{i,j}$ satisfy the following

1. $\mathsf{v}'_{i,j}(t_p) \geq \mathsf{v}_{i,j}(t_p)$ for each reference clock t_p,
2. if there is an active $x > c$ bound from q_i, then $\mathsf{v}'_{i,j}(\widetilde{x}) \leq \mathsf{v}_{i,j}(\widetilde{x})$.
3. if g_j is of the form $x < c$, then $\mathsf{v}'_{i,j}(\widetilde{x}) = \mathsf{v}_{i,j}(t_p)$, where x is a clock of process A_p.

Invariant 7.1.3 guarantees the feasibility of the sequence.

The invariant holds for $j = 0$, as a stronger assumption 7.1.1 holds on these valuations. For $j = 1$, consider δ such that $\mathsf{v}'_{i,1}(t_p) = \mathsf{v}_{i+1}(t_p)$. Then, we have the first part of invariant 7.1.3. The second item continues to hold from the induction hypothesis, as we do not reset any clocks.

For $1 < j \leq n$, the first part of invariant 7.1.3 is immediate. From the definition of locally-relaxed (condition 2), we know that if $x \in LGV(q_{i,j})$, then $x \notin R_1 \cup \cdots \cup R_j$. The second part of invariant 7.1.3 follows as a consequence. For the third part, from the definition of locally-relaxed states (Definition 7.2), we know that if g_j is of the form $x < c$, then $x \in R_1 \cup \cdots \cup R_{j-1}$. Thus, we have the first item of the general invariant 7.1.1.

For the second item, if $x \geq c$ is active in q_{i+1}, then $x \in LGV(q_{i+1})$. From condition 2, we know that $x \notin R_1 \cup \cdots \cup R_l$. Since x is not reset, x had an active lower bound already in q_i, so we can use the induction assumption.

For the third item, if $x \leq c$ is active in q_{i+1}, we have two cases. If $x \in R_1 \cup \cdots \cup R_l$, then $\mathsf{v}'_{i+1}(\widetilde{x}) = \mathsf{v}'_{i+1}(t_p)$. Since we know that $\mathsf{v}_{i+1}(\widetilde{x}) \leq \mathsf{v}_{i+1}(t_p)$ and $\mathsf{v}'_{i+1}(t_p) \geq \mathsf{v}_{i+1}(t_p)$, we get $\mathsf{v}'_{i+1}(\widetilde{x}) \geq \mathsf{v}_{i+1}(\widetilde{x})$. If x is not reset in the

sequence, then x had an active upper bound already in q_i, and so, we use the induction assumption. Thus, invariant 7.1.1 also holds for the case when q_i consists entirely of locally-relaxed states. □

7.2.2 A condition for bounded spread

In this section, we discuss a situation when global-local systems are spread-bounded, but not necessarily 0-spread. The intuition is that if the network synchronizes by executing a global action often, then the spread cannot be too big.

Lemma 7.2. *Suppose that $\mathcal{A}$ is a global-local system, such that there is a bound l on the number of consecutive local actions in a run of $\mathcal{A}$. Then, $\mathcal{A}$ is $l(M+1)$-spread-bounded, where M is the maximum constant used in a guard in $\mathcal{A}$.*

Proof. Pick any run of $\mathcal{A}$ and consider the part of the run consisting of only local actions. We know that such a local run is enclosed between two global actions a_1 and a_2 and is of the following form.

$$\xrightarrow{a_1} (q_{i,0}, \mathsf{v}_{i,0}) \xrightarrow{\delta_1} \xrightarrow[R_1]{g_1} (q_{i,1}, \mathsf{v}_{i,1}) \xrightarrow{\delta_2} \xrightarrow[R_2]{g_2} \dots \xrightarrow{\delta_l} \xrightarrow[R_l]{g_l} (q_{i,l}, \mathsf{v}_{i,l}) \xrightarrow{\delta} (q_{i,l}, \mathsf{v}'_{i,l}) \xrightarrow{a_2} (q_{i+1}, \mathsf{v}_{i+1})$$

Observe that the valuations $\mathsf{v}_{i,0}$ and $\mathsf{v}'_{i,l}$ are synchronized. Consequently, we can talk about duration of this execution, given by $\mathsf{v}_{i+1}(t_p) - \mathsf{v}_{i,0}(t_p)$, where t_p is a reference clock. If the duration is not bigger than lM, then clearly, all the valuations appearing in the sequence are $l(M+1)$-spread-bounded.

If the duration is bigger than lM, then it means that in the local sequence, there is an action taking more than M time. Observe that if an action requires a time elapse of more than M time units, then it can elapse arbitrary time bigger than M. So, if we have such an action, then we can adjust the time elapsed before that action in the interval $(M, +\infty)$. We observe that for any such action, the delay before it can be shortened at will to at most $(M+1)$. In this way, we obtain an execution of duration at most $l(M+1)$.

To summarize, we can modify the sequence so that each action requires a delay of at most $(M+1)$ time units. As the sequence is of length at most l, its duration of execution is at most $l(M+1)$. □

Corollary 7.1. *A global-local system is spread-bounded if no process has a loop containing only local actions.*

7.3. Client-server systems

In this section, we introduce another special class of networks, which we refer to as *client-server systems*. In these networks, we have a server process and several client processes. We denote the server process by S, and the client processes by $C_1, C_2, \cdots, C_k$. These systems can have three kinds of actions:

- local actions of a client, whose domain is $\{C_i\}$ for some i,
- local actions of the server whose domain is $\{S\}$,
- synchronization actions involving a client and the server, whose domain is $\{S, C_i\}$ for some i.

In the context of client-server systems, we will refer to actions that synchronize a client and the server as *communication actions*.

We will now give sufficient conditions for a client-server system to be spread-bounded. We will first focus on conditions for 0-spread-boundedness. Later, we will discuss conditions to conclude that a client-server system is spread-bounded, but not necessarily 0-spread-bounded.

7.3.1 Conditions for 0-spread

In this section, we give some sufficient conditions for a client-server system to be 0-spread. First, we introduce the notion of a tame client.

Definition 7.4. We say that a client C_p is *tame* if no local action of C_p has a clock bound or a reset.

In Lemma 7.3, we will prove the 0-spread-boundedness of client-server systems in which all the clients are tame.

Lemma 7.3. *A client-server system with tame clients is 0-spread.*

Proof. Let S be the server process. Consider a run:

$$(q_0, \mathsf{v}_0) \xrightarrow{\delta_1} \xrightarrow{b_1} (q_1, \mathsf{v}_1) \xrightarrow{\delta_2} \xrightarrow{b_2} \cdots (q_{n-1}, \mathsf{v}_{n-1}) \xrightarrow{\delta_n} \xrightarrow{b_n} (q_n, \mathsf{v}_n)$$

Define v'_i as $\mathsf{v}'_i(t_p) = \mathsf{v}_i(t_s)$ for all reference clocks t_p, and $\mathsf{v}'_i(\widetilde{x}) = \mathsf{v}(\widetilde{x})$ for all other variables. We show that the run with all v_i replaced by v'_i is a 0-spread run. Consider an action in the given run : $(q_i, \mathsf{v}_i) \xrightarrow{\delta_i} \xrightarrow{b_i} (q_{i+1}, \mathsf{v}_{i+1})$. We need to show that $(q_i, \mathsf{v}'_i) \xrightarrow{\delta_i} \xrightarrow{b_i} (q_{i+1}, \mathsf{v}'_{i+1})$ is feasible.

We look at this transition closely.

$$(q_i, \mathsf{v}_i) \xrightarrow{\delta_i} (q_i, \mathsf{v}_i + \delta_i) \xrightarrow{b_i} (q_{i+1}, [R_i](\mathsf{v}_i + \delta_i))$$

We show that

$$(q_i, \mathsf{v}'_i) \xrightarrow{\delta'_i} (q_i, \mathsf{v}'_i + \delta'_i) \xrightarrow{b_i} (q_{i+1}, [R_i](\mathsf{v}'_i + \delta'_i))$$

is possible where δ'_i is a vector consisting of delay $\delta_i(s)$ for each process of the network. In other words, each process does exactly the same delay, the delay executed by the server originally.

If b_i is a local action, by assumption there are no guards in b_i and so, the transition is feasible. If b_i is a synchronization of the client C_p with the server, then $(\mathsf{v}_i+\delta_i)(t_s) = (\mathsf{v}_i+\delta_i)(t_p)$. Moreover, $\mathsf{v}_i(t_s) = \mathsf{v}'_i(t_s)$ by definition of v'_i. This gives $(\mathsf{v}'_i + \delta'_i)(t_p) = (\mathsf{v}'_i + \delta'_i)(t_s) = (\mathsf{v}_i + \delta_i)(t_s) = (\mathsf{v}_i + \delta_i)(t_p)$. So the transition is feasible as the guards of b_i involve only the clocks of C_p and S. □

Next, we propose a more general condition to say that a client-server system is 0-spread. This approach is targeted at client-server systems with identical client processes, which are quite common in the standard literature. In such a setting, we show that if a simpler client-server system, consisting of just one client and the server satisfies a condition, then we can conclude that the original client-server system is 0-spread. We specify the condition that the simpler client-server system should satisfy in Definition 7.5

Definition 7.5. Consider network $\mathcal{A}$ consisting of only two processes: a client and a server. The network is *client-0-spread* if every run in the local time semantics can be made 0-spread by adjusting only the local delays of the client.

Lemma 7.4. *Consider a client-server system with identical client processes. If the network consisting of the server and one client of the network is client-0-spread, then the network with an arbitrary number of clients is 0-spread.*

Proof. Let $\mathcal{N}$ be a client-server system with identical client processes, such that the network consisting of the server and one client of the network is client-0-spread. We will show that we can convert any local run of $\mathcal{N}$ to a 0-spread run.

Consider a local run of $\mathcal{N}$:

$$(q_0, \mathsf{v}_0) \xrightarrow{\delta_1}\xrightarrow{b_1} (q_1, \mathsf{v}_1) \xrightarrow{\delta_2}\xrightarrow{b_2} \cdots (q_{n-1}, \mathsf{v}_{n-1}) \xrightarrow{\delta_n}\xrightarrow{b_n} (q_n, \mathsf{v}_n)$$

Consider a client C_p. Consider the actions in the run belonging to the client C_p and the server. By definition of client-0-spread systems, we know that the reference time of C_p can be made equal to the reference time of the server in each valuation of this run. Further, observe that for all other valuations in the run, the reference time of client C_p does not affect the feasibility of the action, as the action belongs to another process. So, we can keep the reference time of C_p synchronized with the reference time of the server in these valuations. Thus, the reference time of the client C_p is equal to the reference time of the server in all the valuations. We repeat this for all the clients. Note that we can do this any order, since by Lemma 3.4, local actions of different clients can be commuted.

We now make this idea precise. Define v'_i as $\mathsf{v}'_i(t_p) = \mathsf{v}_i(t_s)$ for all reference clocks t_p and $\mathsf{v}'_i(\widetilde{x}) = \mathsf{v}(\widetilde{x})$ for all other variables. The run with all v_i replaced by v'_i is feasible and is a 0-spread run. □

7.3.2 Conditions for bounded spread

In this section, we discuss some situations when client-server systems are spread-bounded, but not necessarily 0-spread. We will present two such cases, one in which there is a bound on time elapsed between two consecutive communications of each client, and the other in which all the timing constraints of the system are of a specific form.

Systems with time bound on consecutive communications

Here we consider client-server systems in which there is a bound on the time elapsed between consecutive communication actions with each client. We will show that such networks are spread-bounded.

Lemma 7.5. *Suppose that $\mathcal{N}$ is a client-server system such that each client communicates with the server every D time units. Then $\mathcal{N}$ is $2D$ spread-bounded.*

Proof. Consider a local run of $\mathcal{N}$.

$$(q_0, \mathsf{v}_0) \xrightarrow{\delta_1}\xrightarrow{a_1} (q_1, \mathsf{v}_1) \xrightarrow{\delta_2}\xrightarrow{a_2} \cdots (q_{n-1}, \mathsf{v}_{n-1}) \xrightarrow{\delta_n}\xrightarrow{a_n} (q_n, \mathsf{v}_n)$$

We show that without any modification this run is $2D$-spread-bounded. Take a position i and consider $|\mathsf{v}_i(t_p) - \mathsf{v}_i(t_q)|$, for arbitrary processes A_p and A_q. We know that $\mathsf{v}_i(t_q)$ is at least as big as the time of the last synchronization of A_q with the server. Similarly, $\mathsf{v}_i(t_p)$ is not bigger than the time of the next synchronization of A_p with the server. So $\mathsf{v}_i(t_q) \geq \mathsf{v}_i(t_s) - D$ and $\mathsf{v}_i(t_p) \leq \mathsf{v}_i(t_s) + D$. □

Systems where the only timing constraints are at wait states

Here, we consider client-server systems in which the timing constraints are only used to model *wait states* where the system needs to elapse a certain amount of time in the state before executing an outgoing action.

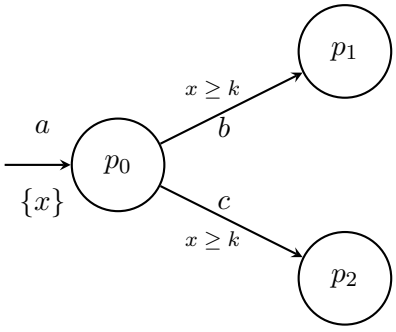


Figure 7.2: Wait states

This is modelled by a reset in all the incoming actions to the state, followed by a lower bound guard of the form $x \geq k$ on all the outgoing actions. We give an example of a wait state in Figure 7.2. We refer to the maximum constant associated to an outgoing action from a wait state as the *wait time* of the state. We now show that in client-server systems where the only timing restrictions come from wait states are spread-bounded.

Lemma 7.6. *Let $\mathcal{N}$ be a client-server system such that all the timing constraints in each process of $\mathcal{N}$ are associated to wait states, and these wait states are not part of any cycle in the respective process. Then $\mathcal{N}$ is spread-bounded, with a bound $N \cdot W$ on the spread, where N is the maximum number of wait states in a process of $\mathcal{N}$, and W is an upper bound on the wait time associated to a wait state in $\mathcal{N}$.*

Proof. Consider a run of $\mathcal{N}$. From this run, we can remove any delay that is not executed from a wait state, since the value of a clock is checked only from a wait state.

In the run thus obtained, observe that the value of any reference clock t_i is such that $0 \leq \mathsf{v}(t_i) \leq N \cdot W$. This implies that the value of the spread at any point in the run is bounded by $N \cdot W$. □

7.4. Deciding 0-spread is PSPACE-complete

In this section, we examine how hard it is to detect if a system is 0-spread. We show that the problem of deciding whether a given network of timed automata is 0-spread is PSPACE-complete. As in the rest of the document, we assume that the given network of timed automata is deterministic: that is, the underlying finite automaton of each component is deterministic.

Let $\mathcal{L}_g(\mathcal{A})$ be the set of all action sequences u such that there exists a local run $(q_0, \mathsf{v}_0) \xrightarrow{u} (q, \mathsf{v})$ where all the valuations are synchronized. In other words, u is a 0-spread run. This also gives a run in the standard global semantics of timed automata. Note that $\mathcal{L}_g(\mathcal{A})$ is prefix-closed.

A regular language L over a finite alphabet and an independence relation is *trace closed* if for every word $u \in L$, all words obtained from u by commuting independent actions (for a definition of independent actions, see Definition 2.48) are also in L.

Lemma 7.7. *A network $\mathcal{A}$ is 0-spread-bounded iff $\mathcal{L}_g(\mathcal{A})$ is trace closed.*

Proof. Take a run $(q_0, \mathsf{v}_0) \xrightarrow{u} (q, \mathsf{v})$ passing only through synchronized valuations. From Lemma 3.5, for every $w \sim u$, automaton $\mathcal{A}$ has a local run $(q_0, \mathsf{v}_0) \xrightarrow{w} (q, \mathsf{v})$. If $\mathcal{A}$ is 0-spread-bounded, then w can be converted to a 0-spread run. Hence $w \in \mathcal{L}_g(\mathcal{A})$.

If $\mathcal{A}$ is not 0-spread-bounded, there exists a local run $(q_0, \mathsf{v}_0) \xrightarrow{u} (q, \mathsf{v})$ which cannot be realized with synchronized valuations. Since the network $\mathcal{A}$ is deterministic, every run on u has to pass through the same sequence of states as in the local run $(q_0, \mathsf{v}_0) \xrightarrow{u} (q, \mathsf{v})$. Hence there can be no other run on u passing through only synchronized valuations. This gives $u \notin \mathcal{L}_g(\mathcal{A})$, and proves the backward direction. □

The following lemma is a general observation about regular languages equipped with an independence relation.

Lemma 7.8. *If a language L is not trace closed then there are, possibly empty, words u, v and two independent letters a, b such that $ubav \in L$ and $uabv \notin L$.*

Proof. Suppose it it not the case. Consider two trace equivalent words $w_1 \sim w_2$ with $w_1 \in L$. By definition of trace equivalence, w_2 is obtained from w_1 by some finite number of permutations of adjacent independent letters. By our assumption $w_2 \in L$. This implies L is trace closed, a contradiction. □

Lemma 7.9. *A language of a deterministic automaton is not trace closed if there are two independent letters a, b, and a reachable state q such that $q \xrightarrow{ab} q_1$, $q \xrightarrow{ba} q_2$ with $L(q_1) \neq L(q_2)$.*

Proof. Let us take u, v and a, b as in Lemma 7.8. The state q is the state reached on u. Then, v is accepted from q_1 but not from q_2. □

The region graph of the network $\mathcal{A}$ contains nodes of the form (q, r) where q is a state and r is a (standard) region (see Section 2.3). There is an edge $(q, r) \xrightarrow{a} (q', r')$ if there exists $v \in r$ and (global) delay $\Delta \in \mathbb{R}_{\geq 0}$ such that $(q, v) \xrightarrow{\Delta, a} (q', \mathsf{v}')$ with $\mathsf{v}' \in r'$. Let $R(\mathcal{A})$ be the region graph of $\mathcal{A}$ seen as an automaton with all states marked accepting. Note that if some action a is not enabled from a node (q, r), then there is no transition on a from state (q, r) of $R(\mathcal{A})$. Therefore $R(\mathcal{A})$ is an incomplete DFA. It can be completed by adding a single sink state.

Lemma 7.10. *$\mathcal{L}_g(\mathcal{A})$ is accepted by DFA $R(\mathcal{A})$. The size of $R(\mathcal{A})$ is singly exponential in the size of $\mathcal{A}$, the transitions from a given state can be computed in* PTIME.

Proposition 7.1. *Checking if a network is 0-spread-bounded can be done in* PSPACE.

Proof. A network $\mathcal{A}$ is not 0-spread-bounded iff $\mathcal{L}_g(\mathcal{A})$ is not trace closed (Lemma 7.7). Consider a deterministic finite automaton $\mathcal{R}(\mathcal{A})$ recognizing $\mathcal{L}_g(\mathcal{A})$ (Lemma 7.10). So, $\mathcal{A}$ is not 0-spread if there exist a, b and q as in Lemma 7.9. Observe that testing $L(q_1) \neq L(q_2)$ can be done in PSPACE as $\mathcal{R}(\mathcal{A})$ is of exponential size and the successor relation can be computed in

PTIME. This gives a PSPACE procedure for checking if $\mathcal{A}$ is not 0-spread. The lemma follows as PSPACE is closed under complement. □

We now turn to showing the lower bound. We give a reduction from the emptiness problem for timed automata to the 0-spread-boundedness problem of a network of timed automata.

Let A be a timed automaton. We construct a network of timed automata $\mathcal{N}$ which consists of processes A', B and C as shown in Figure 7.3. Process A' is obtained from automaton A by adding a self-loop on an action a from each accepting state of A. The action a is a fresh global action (not appearing in A) which synchronizes the processes A', B and C. The process B has a clock x and the process C has a clock y. We reset x in B and y in C on the global action a. There is a local action b in B which has guard $x > k$ and a local action c in C with the guard $y < k$.

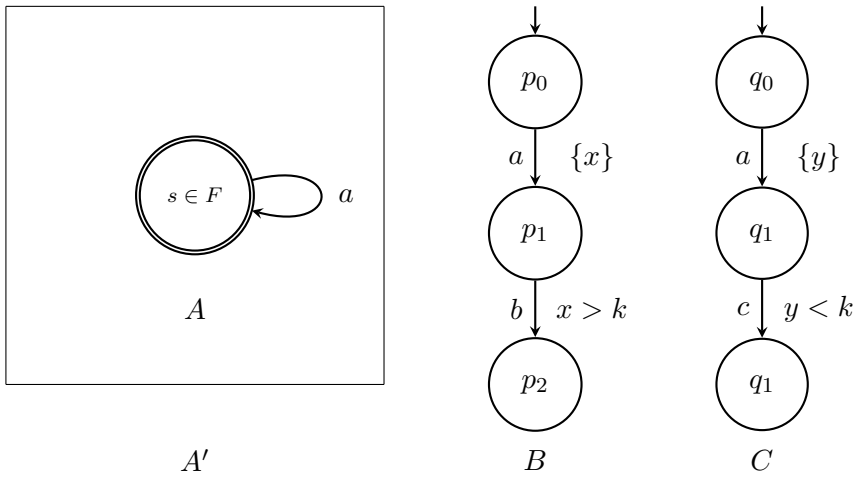


Figure 7.3: Network $\mathcal{N}$

Proposition 7.2. *A is non-empty iff the network $\mathcal{N}$ is not 0-spread-bounded.*

Proof. Suppose that A has an accepting run σ to an accepting state s of A. Note that σ is also a local run of A. Consider this local run σ in $\mathcal{N}$ - from the initial state of A' to the state s. Since every action in σ is a local action of $\mathcal{N}$, this run only passes through synchronized valuations (we can keep the processes B and C synchronized during this run). Now, consider the extension of this run by the execution of actions a, b and c, in that respective order. It may be observed that the valuation obtained after executing b is not a synchronized valuation. Further, it maybe observed that after executing b, no synchronized valuation can execute the action c. Thus, σabc is a local-run of the network, which has non-zero spread. On the other hand if A does not

have an accepting run, a cannot be executed. In this case, all the runs of $\mathcal{N}$ are runs of A. Thus, $\mathcal{N}$ is 0-spread. □

Remark. *Since the network $\mathcal{N}$ considered in the proof of Lemma 7.2 is a global-local system, the* PSPACE *bound also applies to such systems.*

Theorem 7.2. *Checking if a network of timed automata is 0-spread is* PSPACE*-complete.*

7.5. Deciding D-spread is PSPACE-hard

In this section, we investigate the hardness of checking if a system is D-spread. We will show that this problem is PSPACE-hard.

To prove the hardness result, we give a reduction from the emptiness problem for timed automata to D-spread-boundedness problem of a network of timed automata. The proof is very similar to the proof of PSPACE-hardness of the 0-spread-boundedness problem.

Given a timed automaton A, we construct a network of timed automata $\mathcal{N}$ which consists of processes A', B and C as shown in Figure 7.4. Process A' is obtained from automaton A by adding a self-loop on a fresh global action a which synchronizes the processes A', B and C. The clock x of the process B and the clock y of the process C are reset on a. There is a local action b in B which has guard $x > k + D$ and a local action c in C with the guard $y < k$. Observe that the newly constructed network is similar to the network constructed in the 0-spread-boundedness proof (Figure 7.3), except for the fact that the action b has guard $x > k + D$ instead of $x > k$.

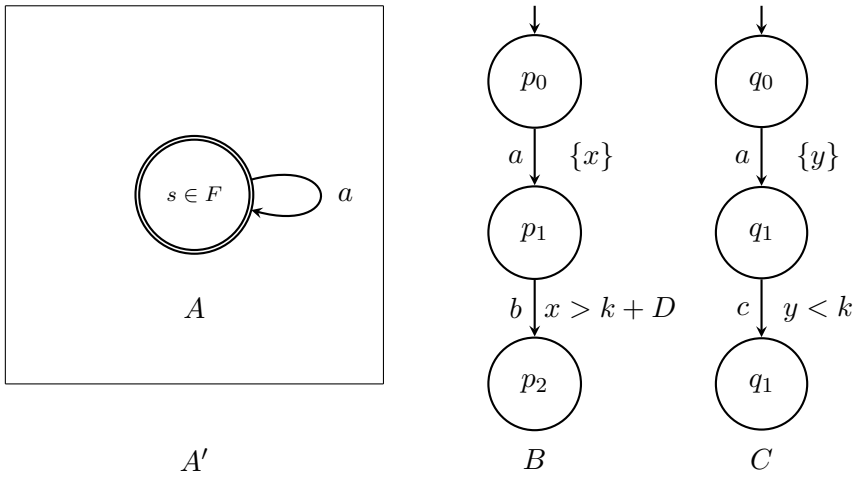


Figure 7.4: Network $\mathcal{N}$

We now state Proposition 7.3 that states that the D-spread-boundedness problem is PSPACE-hard. The proof is similar to the proof of Proposition 7.2 that shows that PSPACE-hardness of the 0-spread-boundedness problem.

Proposition 7.3. *A is non-empty iff the network $\mathcal{N}$ is not D-spread-bounded.*

Theorem 7.3. *Checking if a network of timed automata is D-spread-bounded is* PSPACE*-hard.*

Chapter 8

Implementation of partial order reduction in TChecker

In Chapter 6, we showed that if a network of timed automata is D-spread bounded, then the $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}$ of the network (the transition system obtained by applying $\sqsubseteq^D_M$ subsumption to the local zone graph of the network) can be used to check the reachability of the network. Moreover, using Theorem 6.1, we showed that we can apply a partial order reduction technique based on source sets to the exploration of the $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}$ of a network of timed automata, provided it is D-spread bounded. In this chapter, we work with the assumption that the networks that we consider are spread bounded and that we know a bound D on the spread of these networks. In other words, whenever we consider $\sqsubseteq^D_M$ subsumption for two nodes of the local zone graph of a network, we assume that the network is D-spread bounded.

In this chapter, we discuss an implementation of a partial order reduction procedure for networks of timed automata that is provided in the tool TChecker [HP19]. We discuss two variants of the POR implementation, namely *global-local POR* and *client-server POR*, that are customized and targeted at the two kinds of networks of timed automata that we introduced in Chapter 7, namely, *global-local systems* (see Section 7.2) and *client-server systems* (see Section 7.3), respectively.

Before discussing the implementation, we first discuss the application of an abstract partial order reduction method, namely *reachability-complete set* reduction to a simple transition system in Section 8.1. Equipped with the ideas introduced in this discussion, we then move on to the application of a reachability-complete set reduction method to finite truncations of local zone graphs of networks of timed automata.

In our implementation, we compute a reduction of a finite version of the LZG of a network $\mathcal{N}$, denoted as $\mathsf{POR\text{-}LZG}_r(\mathcal{N})$. Given a network of timed automata $\mathcal{N}$, we consider the local zone graph of $\mathcal{N}$ with the $\sqsubseteq^D_M$ subsumption between its nodes. We first augment $\mathsf{LZG}(\mathcal{N})$ by storing some

additional information in its nodes to get the *POR-zone graph* of $\mathcal{N}$ (denoted as $\mathsf{POR\text{-}LZG}(\mathcal{N})$). This additional information is crucial in computing the subset of actions that are to be explored from a node in the restricted transition system that we want to obtain. Thus, effectively, $\mathsf{POR\text{-}LZG}(\mathcal{N})$ can be viewed as $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}(\mathcal{N})$, with each node augmented with some additional information. Further, our reduced transition system, $\mathsf{POR\text{-}LZG}_r(\mathcal{N})$, can be viewed as the part of $\mathsf{POR\text{-}LZG}(\mathcal{N})$ that is reachable from the initial node of $\mathsf{POR\text{-}LZG}(\mathcal{N})$ using the reduced set of actions proposed by our implementation.

In this chapter, we present a general POR-reachability algorithm (Algorithm 5) that takes three inputs, namely a network $\mathcal{N}$ of timed automata, a subroutine Next that gives the successor relation of $\mathsf{POR\text{-}LZG}_r(\mathcal{N})$, and a node covering relation $\sqsubseteq$ between the nodes of $\mathsf{POR\text{-}LZG}(\mathcal{N})$. The implementation of the procedure Next and the node covering relation $\sqsubseteq$ depends on the POR method that we use. We will propose two different implementations of Next and the node-covering relation $\sqsubseteq$, one each for global-local systems and client server systems. Finally, in Section 8.5, we propose optimizations to the Next procedures that exploit specific properties of global-local and client-server systems, respectively.

8.1. Partial order reduction for transition systems

In this section, we discuss the application of partial order reduction to a simple untimed transition system. We introduce the notion of a *reachability-complete set* for a transition system and define the idea of a *reduced transition system* that is obtained by the application of a reachability-complete set reduction to a transition system. We then prove that the reduced transition system is sound and complete with respect to reachability. These ideas will be useful when we discuss the application of partial order reduction to the $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}$ of a D-spread bounded network of timed automata.

We first recall the definitions of a transition system and recall the notion of a run of the transition system.

Definition 8.1 (Transition system)**.** A *transition system* is a tuple $\langle S, \Sigma, s_0, \rightarrow, F\rangle$, where S is a set of states, Σ is a finite alphabet of actions, $\rightarrow \subseteq S \times \Sigma \times S$ is a transition relation between states, $s_0 \in S$ is an initial state and $F \subseteq S$ is a set of accepting states. We write $s \xrightarrow{a} s'$ to denote that $(s, a, s') \in \rightarrow$. We sometimes refer to $\rightarrow$ as *successor relation* of the transition system.

Definition 8.2 (Run of the transition system)**.** A *run* in the transition system T from a state s is a sequence of transitions starting in s: $s \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \cdots \xrightarrow{a_n} s_n$. We denote it by $s \xrightarrow{\sigma} s_n$ where $\sigma = a_1 \cdot a_2 \cdots a_n$ is a sequence of actions.

The reachability problem for a transition system T asks if there is a run of T from its initial state to an accepting state.

We now introduce the notion of a simulation relation over a transition system.

Definition 8.3 (Simulation relation)**.** Given a transition system T, we say that $\preceq \subseteq S \times S$ is a *simulation relation* over T if for all $s_1 \preceq s_2$, $s_1 \xrightarrow{a} s_1'$ implies that there exists a s_2' such that $s_2 \xrightarrow{a} s_2'$, and $s_1' \preceq s_2'$. We say s' simulates s if $s \preceq s'$.

Lemma 8.1. *Given a transition system $T = \langle S, \Sigma, s_0, \rightarrow, F\rangle$ and a simulation relation $\preceq$ over T, if $s \preceq s'$ and $s \xrightarrow{\sigma} s_n$, then $s' \xrightarrow{\sigma} s_n'$, for some s_n' such that $s_n \preceq s_n'$.*

Proof. Let σ be a run from s of the following form

$$s \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots s_{n-1} \xrightarrow{a_n} s_n.$$

We will show that there is a run of the form

$$s' \xrightarrow{a_1} s_1' \xrightarrow{a_2} \cdots s_{n-1}' \xrightarrow{a_n} s_n'$$

such that $s_i \preceq s_i'$, for $1 \leq i \leq n$.

The proof follows by induction on n, the number of transitions in σ. The base case where $n = 0$ follows trivially, since we have $s \preceq s'$.

Consider the induction step. By induction hypothesis, we know that there is a run σ' of the form

$$s' \xrightarrow{a_1} s_1' \xrightarrow{a_2} \cdots \xrightarrow{a_k} s_k'$$

such that $s_i \preceq s_i'$ for $1 \leq i \leq k$. From the run σ, we know that $s_k \xrightarrow{a_{k+1}} s_{k+1}$. Further, since $\preceq$ is a simulation relation and $s_k \preceq s_k'$, we know that there exists a s_{k+1}' such that $s_k' \xrightarrow{a_{k+1}} s_{k+1}'$ and $s_{k+1} \preceq s_{k+1}'$. Therefore, the run σ' that we have constructed can be extended by the transition a_{k+1} to get the run

$$s' \xrightarrow{a_1} s_1' \xrightarrow{a_2} \cdots \xrightarrow{a_k} s_k' \xrightarrow{a_{k+1}} s_{k+1}'$$

where $s_i \preceq s_i'$ for $1 \leq i \leq k+1$. □

Thanks to Lemma 8.1, we can propose Algorithm 4 to check the reachability of a transition system. The procedure takes two inputs, a transition system $T = \langle S, \Sigma, s_0, \rightarrow, F\rangle$, and a simulation relation $\preceq$ over T, and checks if an accepting state is reachable in T.

Lemma 8.2. *If a state s is in the list* Waiting *of Algorithm 4, then s is reachable in T.*

Algorithm 4 Reachability algorithm

Input: A transition system $T = \langle S, \Sigma, s_0, \rightarrow, F \rangle$ and a simulation relation $\preceq$ over T.

Output: true iff accepting state reachable in T.

```
1:  Set Waiting = Visited := {s_0}
2:  if s_0 is accepting then return true
3:  while Waiting ≠ ∅ do
4:      s = pop(Waiting)
5:      for all s' s.t s -a-> s' do
6:          if s' is accepting then
7:              return true
8:          else if ∃ s'' ∈ Visited s.t. s' ≼ s'' then
9:              Skip
10:         else
11:             for s'' ∈ Visited do
12:                 if s'' ≼ s' then
13:                     Remove s'' from Visited and Waiting
14:             Add s' to Waiting and Visited
15: return false
```

Proof. We show that the statement of the lemma is a loop invariant of Algorithm 4.

Before the first iteration of the loop, we know that Waiting $= \{s_0\}$. Since s_0 is vacuously reachable in T, the statement is true at the beginning of the loop.

Now, assume that the loop invariant holds before an iteration of the loop (line 4.) Thus, we know that each state in the list Waiting at this point is reachable in T. In the iteration of the loop, we pick a state s from Waiting and add some states s' to Waiting such that $s \xrightarrow{a} s'$ for some $a \in \Sigma$. Consider such a state s' that is added to Waiting in this iteration of the loop. Since we know that s is reachable in T, we have a run $s_0 \xrightarrow{\sigma} s$ in T. We can extend this run by a to get $s_0 \xrightarrow{\sigma} s \xrightarrow{a} s'$. Then, it follows that s' is also reachable in T. Thus, we know that if a state s' is added to Waiting in this iteration of the loop, then s' is reachable in T. Therefore, the statement of the lemma continues to hold at the end of the loop. □

Lemma 8.3. ***(Soundness)*** *If Algorithm 4 returns* true*, then T has an accepting run.*

Proof. We know that Algorithm 4 can return true by either executing line 2 or line 7. Line 2 is only executed in the special case when the initial state is an accepting state. In this case, we know that the claim is vacuously true. Next, consider the case when line 7 is executed. We can infer from the

algorithm that this happens only when a state s has has just been popped from Waiting and there exists a $s \xrightarrow{a} s'$ such that s' is accepting. Since s has just been popped from Waiting, from Lemma 8.2, we know that s is reachable in T. It follows that s' is a reachable accepting state in T. Hence, T has an accepting run. □

Lemma 8.4. *If a state s is added to* Visited *at some step of Algorithm 4, then at every step afterwards there is a state s' in* Visited *such that $s \preceq s'$.*

Proof. Let s be a state which has been added to Visited at some step of Algorithm 4. If s is in Visited till the termination of the algorithm, then we are done. Suppose not. The only step of the algorithm which removes a state s from Visited is line 13. But we can see that in this case we have $s' \in$ Visited and $s \preceq s'$ because of the guard before line 13. Further, if s' is removed at a later point in the algorithm, then we know that another state s'' such that $s' \preceq s''$ is added to Visited. Then, s'' serves as the candidate state in Visited that covers s. □

Lemma 8.5. *Let σ be run of T of the form*

$$s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_{n-1}} s_{n-1} \xrightarrow{a_n} s_n$$

If the algorithm does not return true*, then at the termination of the algorithm for every $0 \leq i \leq n$, there exists a state s'_i in the set* Visited *such that $s_i \preceq s'_i$.*

Proof. Suppose that the algorithm does not return true. The proof follows by induction on the number of transitions in σ.

For the base case, we know that the state s_0 is added to Visited in line 1 of the algorithm. By Lemma 8.4, till the end of the execution there will be some $s'_0 \in$ Visited with $s_0 \preceq s'_0$.

By induction hypothesis, assume that there exists a state s'_i in Visited, such that $s_i \preceq s'_i$. Assume that it s'_i is the $\preceq$-biggest such state. To be in Visited, s'_i must also have been in Waiting at some stage. Since when the algorithm terminates the list Waiting is empty, at some moment s'_i must have been popped from Waiting.

From the run in T, we know that $s_i \xrightarrow{a_{i+1}} s_{i+1}$. Since $\preceq$ is a simulation relation in T and $s_i \preceq s'_i$, we have $s'_i \xrightarrow{a_{i+1}} s'_{i+1}$, where $s_{i+1} \preceq s'_{i+1}$. If s'_{i+1} is added to Visited, then by Lemma 8.4, we are done. If s'_{i+1} is not added to Visited, then there are two possibilities:

- s'_{i+1} is an accepting state. In this case, Algorithm 4 terminates by returning true, and we have assumed that it is not the case.
- There exists a state s''_{i+1} in Visited such that $s'_{i+1} \preceq s''_{i+1}$. But in this case, we have $s_{i+1} \preceq s'_{i+1} \preceq s''_{i+1}$. By transitivity of $\preceq$ relation, $s_{i+1} \preceq s''_{i+1}$. Thus, in this case we have s''_{i+1} in Visited, such that $s_{i+1} \preceq s''_{i+1}$.

□

Corollary 8.1. ***(Completeness)*** *If T has a reachable accepting state, then Algorithm 4 returns* true*.*

Proof. Clearly the algorithm terminates, as every state is added to the list Waiting at most once. By Lemma 8.5, if there is a run to an accepting state s, then s must be added to Visited at some stage. But this is impossible because no accepting state can be added to Visited (with the exception of the case when s_0 is an accepting state; In this case however, the algorithm terminates at line 2.).

□

From Lemma 8.3 and Corollary 8.1, we have Theorem 8.1, that gives the correctness of Algorithm 4.

Theorem 8.1. *Algorithm 4 returns* true *if and only if T has a reachable accepting state.*

Reachability-complete set reduction

In this section, we discuss the application of a partial order reduction technique to the exploration of a transition system. We first introduce the notion of a *reduced transition system* generated by a subset of the successor relation.

Definition 8.4 (Reduced transition system)**.** Let $T = \langle S, \Sigma, s_0, \rightarrow, F\rangle$ be a transition system and let $\rightarrow_r \subseteq \rightarrow$ be a subset of the successor relation of T. We then define the reduced transition system induced by $\rightarrow_r$, denoted by T_r, as the restriction of T to the states reachable from s_0 using only transitions in $\rightarrow_r$. Formally, T_r is given by the tuple $\langle S_r, \Sigma, s_0, \rightarrow_r, F_r\rangle$, where S_r is the smallest set such that $s_0 \in S_r$, and if $s \in S_r$ and $s \xrightarrow{a}_r s'$ for some $a \in \Sigma$, then $s' \in S_r$. The set of accepting states of T_r is given by $F_r = S_r \cap F$.

Next, we specify a property that the subset of actions should satisfy for the reduced transition system to be sound and complete with respect to reachability.

Definition 8.5 (*Reachability-complete set*)**.** Consider a transition system $T = \langle S, \Sigma, s_0, \rightarrow, F\rangle$, a transition relation $\rightarrow_r \subseteq \rightarrow$ and the reduced transition system T_r induced by $\rightarrow_r$. We say that $\rightarrow_r$ is *reachability-complete* for T, if T_r contains an accepting state whenever an accepting state is reachable from s_0 in T.

Observe that if no accepting state is reachable in T, then the above definition puts no restrictions on T_r. Otherwise we require that in T_r remains at least one of the reachable accepting states.

Corollary 8.2. *Suppose that $T = \langle S, \Sigma, s_0, \rightarrow, F\rangle$ is a transition system, and $\rightarrow_r \subset \rightarrow$ that is reachability-complete for T. For T_r the reduced transition system induced by $\rightarrow_r$: T_r has an accepting run if and only if T has an accepting run.*

8.2. Implementation of partial order reduction in TChecker

In this section, we explain the implementation of partial order reduction in TChecker. We consider two kinds of networks of timed automata, referred to as *global-local systems* and *client-server systems*, and propose two respective variants of the implementation - *global-local POR* (Section 8.3) and *client-server POR* (Section 8.4).

Recall that the standard reachability algorithm (Algorithm 4) takes a transition system $T = \langle S, \Sigma, s_0, \rightarrow, F\rangle$ and solves the reachability problem on T. We have also remarked that if we are only interested in reachability, then we could take a smaller successor relation $\rightarrow_r$ and explore the reduced transition system T_r provided it is reachability-complete. The difficulty is to find $\rightarrow_r$ without completely exploring T in advance.

We are interested in the local zone graph (LZG) of a network of timed automata. We will use the structure of this transition system to compute $\rightarrow_r$ on the fly. But we cannot do this directly. For this purpose, we store some additional information in the nodes of LZG, which helps in computing the subset of actions that are to be explored from that node in the restricted transition system that we want to obtain. This augmented LZG is referred to as the POR-zone graph, denoted by POR-LZG. Thus, POR-LZG can be viewed as LZG, with each node augmented with some additional information. Further, the transition system POR-LZG$_r$ can be viewed as the part of POR-LZG (and therefore the LZG) that can be reached using actions in $\rightarrow_r$ from the initial node of the POR-LZG.

We now describe the POR-zone graph in detail. Let $\mathcal{N}$ be a network of timed automata and let LZG($\mathcal{N}$) be the local zone graph of $\mathcal{N}$. As seen earlier in Section 4.3, the nodes of LZG($\mathcal{N}$) are of the form (q, Z) where q is a state of $\mathcal{N}$ and Z is a local zone. The POR-zone graph of $\mathcal{N}$, denoted as POR-LZG($\mathcal{N}$), is a transition system whose nodes are nodes of LZG($\mathcal{N}$) annotated by an additional parameter called the *rank* of that node, which helps in deciding the set of successors to be explored from that node. Thus, a node of POR-LZG($\mathcal{N}$) is of the form (q, Z, r) where (q, Z) is a node of LZG($\mathcal{N}$) and $r \in \{0, \ldots, n\}$, where n is the number of processes in the network $\mathcal{N}$. The transition relation of POR-LZG($\mathcal{N}$) is defined as follows: if $(q, \mathsf{Z}) \xrightarrow{a} (q', \mathsf{Z}')$ in LZG($\mathcal{N}$), and (q, Z, r) is a node of POR-LZG($\mathcal{N}$), then $(q, \mathsf{Z}, r) \xrightarrow{a} (q', \mathsf{Z}', r')$, where $r' \in \{0, \ldots, n\}$. In the sequel we will choose particular values of r', but this will depend on the method that we will use.

Before going further, we state a property of POR-LZGs that follows immediately from its definition and Corollary 4.1.

Lemma 8.6. *If* $(q, \mathsf{Z}, r) \xrightarrow{u} (q', \mathsf{Z}', r')$ *and* $u \sim w$, *then* $(q, \mathsf{Z}, r) \xrightarrow{w} (q', \mathsf{Z}', r'')$, *for arbitrary* r''.

We now present a general POR-reachability algorithm (Algorithm 5). It is a parametrization of the standard reachability algorithm for generic transition systems (Algorithm 4) with a function calculating the transition relation and the covering relation. The POR-reachability procedure takes three inputs:

1. a network $\mathcal{N}$ of timed automata.
2. a subroutine Next that takes as input a node (q, Z, r) of POR-LZG($\mathcal{N}$) and returns a set of successors $\{(q_1, \mathsf{Z}_1, r_1), \ldots, (q_k, \mathsf{Z}_k, r_k)\}$ such that $(q, \mathsf{Z}) \to (q_i, \mathsf{Z}_i)$ in LZG($\mathcal{N}$) for each $0 \leq i \leq k$.
3. a node covering relation $\sqsubseteq$ between nodes of the POR-LZG($\mathcal{N}$).

The implementation of the procedure Next that computes the successors of a given node in the POR-zone graph depends on the POR method that we use. We will propose two different implementations: one for global-local systems, called $\mathsf{Next}_{\mathsf{gl}}$ presented in Algorithm 6, and one for client server systems, called $\mathsf{Next}_{\mathsf{cs}}$ presented in Algorithm 8. The details about the exact mechanism of how the rank determines the set of successors from a node is explained in the descriptions of $\mathsf{Next}_{\mathsf{gl}}$ and $\mathsf{Next}_{\mathsf{cs}}$ algorithms given in Sections 8.3 and 8.4.

A transition relation $\to_r$ determines Next in a direct way:

$$(q', \mathsf{Z}', r') \in \mathsf{Next}(q, \mathsf{Z}, r) \text{ when } (q, \mathsf{Z}, r) \xrightarrow{a}_r (q', \mathsf{Z}', r') \text{ for some } a.$$

In later sections, we will introduce $\to_{\mathsf{gl}}$ determining $\mathsf{Next}_{\mathsf{gl}}$, and $\to_{\mathsf{cs}}$ determining $\mathsf{Next}_{\mathsf{cs}}$.

Likewise, the node covering relation $\sqsubseteq$ also depends on the POR method that we use. The node-covering relation for global-local systems and client-server systems are defined in Sections 8.3 and 8.4, respectively.

Lemma 8.7. *Suppose that* POR-LZG$_r(\mathcal{N})$ *has a path to a node* (q, Z, r). *Then,* $\mathcal{N}$ *has a local run* $(q_0, \mathsf{v}_0) \to (q, \mathsf{v})$, *for some* $\mathsf{v} \in \mathsf{Z}$.

Proof. Let $(q_0, \mathsf{Z}_0, 0) \xrightarrow{\sigma} (q, \mathsf{Z}, r)$ be a path in POR-LZG$_r(\mathcal{N})$. From the definition of POR-LZG$_r$, this implies that there is a path in LZG($\mathcal{N}$) of the form $(q_0, \mathsf{Z}_0) \xrightarrow{\sigma} (q, \mathsf{Z})$. Then, from the soundness of the local-zone graph given by Theorem 4.1, we know that $\mathcal{N}$ has a run of the form $(q_0, \mathsf{v}_0) \to (q, \mathsf{v})$, for some $\mathsf{v} \in \mathsf{Z}$. □

Algorithm 5 POR reachability algorithm

Input: A network of timed automata $\mathcal{N}$, a procedure $\mathsf{Next} : (Q \times Z \times \mathbb{N}) \mapsto 2^{(Q \times Z \times \mathbb{N})}$, and a node covering relation $\sqsubseteq$.

Output: true iff accepting state reachable in $\mathsf{ZG}(\mathcal{N})$.

```
1: Set Waiting = Visited := {(q_0, Z_0, 0)}
2: if q_0 is accepting then return true
3: while Waiting ≠ ∅ do
4:     (q, Z, r) = pop(Waiting)
5:     for all (q', Z', r') ∈ Next(q, Z, r) do
6:         if q' is accepting then
7:             return true
8:         else if ∃ (q'', Z'', r'') ∈ Visited s.t. (q', Z', r') ⊑ (q'', Z'', r'') then
9:             Skip
10:        else
11:            for (q'', Z'', r'') ∈ Visited do
12:                if (q'', Z'', r'') ⊑ (q', Z', r') then
13:                    Remove (q'', Z'', r'') from Visited and Waiting
14:            Add (q', Z', r') to Waiting and Visited
15: return false
```

Lemma 8.8. *Suppose that $\mathcal{N}$ has a run to an accepting state. Then, if $\rightarrow_r$ is complete for reachability for $\mathsf{POR\text{-}LZG}(\mathcal{N})$, then there is a path to an accepting node in $\mathsf{POR\text{-}LZG}_r(\mathcal{N})$.*

Proof. Consider a run of $\mathcal{N}$ of the form

$$(q_0, \mathsf{v}_0) \xrightarrow{\delta_1} \xrightarrow{a_1} (p_1, \mathsf{v}_1) \cdots \xrightarrow{a_{n-1}} (p_{n-1}, \mathsf{v}_{n-1}) \xrightarrow{\delta_n} \xrightarrow{a_n} \xrightarrow{\delta_{n+1}} (p_n, \mathsf{v}_n)$$

where q_n is an accepting state of $\mathcal{N}$.

By completeness of LZG, there exists a path in $\mathsf{LZG}(\mathcal{N})$ of the form

$$(q_0, \mathsf{Z}_0) \xrightarrow{a_1} (p_1, \hat{\mathsf{Z}}_1) \xrightarrow{a_2} \cdots (p_{n-1}, \hat{\mathsf{Z}}_{n-1}) \xrightarrow{a_n} (p_n, \hat{\mathsf{Z}}_n)$$

such that $\mathsf{v}_0 \in \mathsf{Z}_0$, $\mathsf{v}_n \in \hat{\mathsf{Z}}_n$ and $\mathsf{v}_i \in \hat{\mathsf{Z}}_i$, for each $i \in \{1, \cdots, n-1\}$.

By definition of $\mathsf{POR\text{-}LZG}(\mathcal{N})$, we have a path in $\mathsf{POR\text{-}LZG}(\mathcal{N})$ of the following form

$$(q_0, \mathsf{Z}_0, r_0) \xrightarrow{a_1} (p_1, \hat{\mathsf{Z}}_1, r'_1) \xrightarrow{a_2} \cdots (p_{n-1}, \hat{\mathsf{Z}}_{n-1}, r'_{n-1}) \xrightarrow{a_n} (p_n, \hat{\mathsf{Z}}_n, r'_n).$$

Since $\rightarrow_r$ is complete for reachability for $\mathsf{POR\text{-}LZG}(\mathcal{N})$, we have a path of the following form

$$(q_0, \mathsf{Z}_0, r_0) \xrightarrow{\beta_1}_r (q_1, \mathsf{Z}_1, r_1) \xrightarrow{\beta_2} \cdots (q_{l-1}, \mathsf{Z}_{l-1}, r_{l-1}) \xrightarrow{\beta_l} (q_l, \mathsf{Z}_l, r_l).$$

where q_l is an accepting state of $\mathcal{N}$. This path is an accepting path in $\mathsf{POR\text{-}LZG}_r(\mathcal{N})$ □

Lemma 8.9. *If a node* (q, Z, r) *is in the list* Waiting *of Algorithm 5, then* $\mathcal{N}$ *has a local run* (q_0, v_0) *to* (q, v)*, for some valuation* $\mathsf{v} \in \mathsf{Z}$*.*

Proof. We will prove that the following statement is a loop invariant of Algorithm 5: If a node (q, Z, r) is in the list Waiting of Algorithm 5, then (q, Z, r) is reachable in $\mathsf{POR\text{-}LZG}_r(\mathcal{N})$. Then, using the soundness of $\mathsf{POR\text{-}LZG}_r(\mathcal{N})$ as given by Lemma 8.7, we can conclude that there is a local run in $\mathcal{N}$ of the form $(q_0, \mathsf{v}_0) \to (q, \mathsf{v})$, for some $\mathsf{v} \in \mathsf{Z}$.

The proof of the loop invariant follows as in the proof of Lemma 8.2 with $\mathsf{POR\text{-}LZG}_r(\mathcal{N})$ as the transition system.

□

The proofs of soundness (Lemma 8.10) and completeness of (Lemma 8.11) of Algorithm 5 follow similarly to the proofs of Lemma 8.3 and Corollary 8.1, that respectively gives the soundness and completeness of the standard reachability algorithm, Algorithm 4.

Lemma 8.10. ***(Soundness)*** *If Algorithm 5 returns* true*, then* $\mathcal{N}$ *has an accepting run.*

Lemma 8.11. ***(Completeness)*** *If* $\mathsf{POR\text{-}LZG}(\mathcal{N})$ *has a reachable accepting node, then Algorithm 4 returns* true*.*

Remark. *When we write reachable in Lemma 8.11, we mean "reachable with a synchronized valuation". However, recall that we consider timed automata without state invariants and time-elapsed zones. This means that every reachable local zone contains a synchronized valuation.*

Using Lemma 8.10 and Lemma 8.11, we can now state Theorem 8.2.

Theorem 8.2. *Let* $\mathcal{N}$ *be a network of timed automata. Suppose that* $\to_r$ *is complete for reachability for* $\mathsf{POR\text{-}LZG}(\mathcal{N})$*. Suppose that* Next *is determined by* $\to_r$*, and* $\sqsubseteq$ *is a simulation relation w.r.t.* $\to_r$*. Algorithm 5 returns* true *if and only if an accepting state is reachable in* $\mathsf{POR\text{-}LZG}_r(\mathcal{N})$*.*

Proof. Lemma 8.10 gives us the forward direction of the theorem that says that if Algorithm 5 returns true, then $\mathcal{N}$ has a reachable accepting state.

The reverse direction is given by Lemma 8.8 and Corollary 8.11. □

In Section 8.3 and Section 8.4, we will propose Next functions and $\sqsubseteq$ relations for different kinds of networks of timed automata. We will use the above theorem to show that the instance of Algorithm 5 is correct for those functions.

8.3. Global-local POR

In this section, we propose a POR implementation that is customized and targeted at a class of network of timed automata called *global-local systems*. We describe a procedure $\mathsf{Next}_{\mathsf{gl}}$ that computes the successors of a node of the POR-zone graph and a covering relation $\sqsubseteq_{\mathsf{gl}}$ for the nodes of the POR-zone graph of a global-local system.

8.3.1 Global-local systems

We first briefly recall the properties of global-local systems discussed in detail in Section 7.2. A global-local system $\mathcal{N} = \{A_1, A_2, \cdots, A_n\}$ satisfies two properties:

(gl-trans) $\mathcal{N}$ has only two kinds of actions - *local actions* whose domain is $\{A_i\}$ for some i, and *global actions* whose domain is $\{A_1, A_2, \cdots, A_n\}$.

(gl-final) Every accepting state of $\mathcal{N}$ is only reachable through a global action. If a tuple $(q_1, \cdots, q_n)$ is an accepting state, then for $1 \leq i \leq n$ each incoming transition to q_i is labelled by a global action.

Remark. *Suppose that $\mathcal{N}$ is a network satisfying the condition (gl-trans). Then, $\mathcal{N}$ can be transformed to a global-local system by introducing a dummy accepting state in each process, and adding transitions labelled by a new global action from the (original) accepting states to this dummy accepting state.*

Successor computation for global-local systems

The idea of $\rightarrow_{\mathsf{gl}}$ is to make processes move in turns: first, we move the first process, then the second, etc. until a global action occurs that restarts this mechanism. This is where we use the third component r - it records the process which executed the last action. Given a node (q, Z, r) of the $\mathsf{POR\text{-}LZG}$ of a global-local system, only actions involving processes A_i for $i \geq r$ are proposed by $\mathsf{Next}_{\mathsf{gl}}$.

Definition 8.6. We define a subset $\rightarrow_{\mathsf{gl}}$ of $\rightarrow$ transitions. Given a node (q, Z, r) of $\mathsf{POR\text{-}LZG}$ and a transition $(q, \mathsf{Z}) \xrightarrow{a} (q', \mathsf{Z}')$ of LZG, we have $(q, \mathsf{Z}, r) \xrightarrow{a}_{\mathsf{gl}} (q', \mathsf{Z}', r')$ if either: (i) a is a local action of a process $A_{r'}$ for some $r' \geq r$; or (ii) a is a global action and $r' = 0$.

The associated function $\mathsf{Next}_{\mathsf{gl}}$ computing the set of $\rightarrow_{\mathsf{gl}}$ successors of node is presented in Algorithm 6.

Lemma 8.12. *The successor relation $\rightarrow_{\mathsf{gl}}$ is complete for reachability for transition systems of global-local systems.*

Algorithm 6 $\mathsf{Next}_{\mathsf{gl}}(q, \mathsf{Z}, r)$

Input: A node (q, Z, r) of the POR-zone graph of a global-local network of timed automata A
Output: a set Source of successors of (q, Z, r).

```
1: Source := ∅
2: for every transition (q, Z) -a-> (q', Z') do
3:     if a is a global action then
4:         Add (q', Z', 0) to Source
5:     else
6:         {r'} = dom(a)
7:         if (r' ≥ r) then
8:             Add (q', Z', r') to Source
9: return Source
```

Proof. Suppose that $\mathcal{N}$ is a global-local system. Let $\mathsf{POR\text{-}LZG}_{\mathsf{gl}}(\mathcal{N})$ be the reduced transition system of $\mathsf{POR\text{-}LZG}(\mathcal{N})$ induced by $\rightarrow_{\mathsf{gl}}$. We show that the reduced system is reachability complete. Consider a path:

$$(q_0, \mathsf{Z}_0, 0) \xrightarrow{\sigma_1} (q_1, \mathsf{Z}_1, r_1) \xrightarrow{g_1} (q_2, \mathsf{Z}_2, 0) \xrightarrow{\sigma_2} (q_3, \mathsf{Z}_3, r_3) \xrightarrow{g_2} (q_4, \mathsf{Z}_4, 0) \cdots$$
$$(q_{2n-2}, \mathsf{Z}_{2n-2}, 0) \xrightarrow{\sigma_n} (q_{2n-1}, \mathsf{Z}_{2n-1}, r_{2n-1}) \xrightarrow{g_n} (q_{2n}, \mathsf{Z}_{2n}, 0)$$

in $\mathsf{POR\text{-}LZG}(\mathcal{N})$, where σ_i's are sequences of local actions and g_i's are global actions. Observe that after executing a global action, the rank, i.e., the third component of a configuration is 0.

It is enough to show that there are $r'_1, \ldots, r'_{2n-1}$ and $\sigma'_1, \ldots, \sigma'_n$ such that a path of the form

$$(q_0, \mathsf{Z}_0, 0) \xrightarrow{\sigma'_1}_{\mathsf{gl}} (q_1, \mathsf{Z}_1, r'_1) \xrightarrow{g_1}_{\mathsf{gl}} (q_2, \mathsf{Z}_2, 0) \xrightarrow{\sigma'_2}_{\mathsf{gl}} (q_3, \mathsf{Z}_3, r'_3) \xrightarrow{g_2}_{\mathsf{gl}} (q_4, \mathsf{Z}_4, 0) \cdots$$
$$(q_{2n-2}, \mathsf{Z}_{2n-2}, 0) \xrightarrow{\sigma'_n}_{\mathsf{gl}} (q_{2n-1}, \mathsf{Z}_{2n-1}, r'_{2n-1}) \xrightarrow{g_n}_{\mathsf{gl}} (q_{2n}, \mathsf{Z}_{2n}, 0)$$

exists in $\mathsf{POR\text{-}LZG}_{\mathsf{gl}}(\mathcal{N})$. We will additionally ensure that $\sigma'_i \sim \sigma_i$.

Firstly, we know that the initial node of $\mathsf{POR\text{-}LZG}(\mathcal{N})$ has rank 0. Further, recall that, if a global action is feasible from a node of $\mathsf{POR\text{-}LZG}(\mathcal{N})$, then it is allowed by $\rightarrow_{\mathsf{gl}}$ (irrespective of the rank of the node).

Next, consider a segment consisting of only local actions in the aforementioned path in $\mathsf{POR\text{-}LZG}(\mathcal{N})$:

$$(q_{i_0}, \mathsf{Z}_{i_0}, r_{i_0}) \xrightarrow{a_1} (q_{i_1}, \mathsf{Z}_{i_1}, r_{i_1}) \xrightarrow{a_2} (q_{i_2}, \mathsf{Z}_{i_2}, r_{i_2}) \cdots \xrightarrow{a_l} (q_{i_l}, \mathsf{Z}_{i_l}, r_{i_l}).$$

Each action in this sequence involves only one process. By Lemma 8.6, we can commute actions of different processes in this sequence, to get an interleaving resulting in the same final node, $(q_{i_l}, \mathsf{Z}_{i_l}, r'_{i_l})$, except maybe for the last component of the triple. For every process A_j, consider the subsequence σ^j

of the above sequence consisting only of actions of process j. Since there may be no actions of some process in some segment, we consider only non-empty sequences: $\sigma^{j_1}, \sigma^{j_2}, \ldots, \sigma^{j_k}$ for some $1 \leq j_1 < j_2 < \cdots < j_k \leq n$. Recalling the definition of $\rightarrow_{\mathsf{gl}}$, we see that in $\mathsf{POR\text{-}LZG}_{\mathsf{gl}}(\mathcal{N})$, we have a path

$$(q_{i_0}, \mathsf{Z}_{i_0}, 0) \xrightarrow{\sigma^{j_1}}_{\mathsf{gl}} (q'_{j_1}, \mathsf{Z}'_{j_1}, j_1) \xrightarrow{\sigma^{j_2}}_{\mathsf{gl}} (q'_{j_2}, \mathsf{Z}'_{j_2}, j_2) \cdots \xrightarrow{\sigma^{j_k}}_{\mathsf{gl}} (q_{j_k}, \mathsf{Z}_{j_k}, j_k) \ .$$

This shows how, for all i, we can obtain σ'_i from σ_i, and this completes the proof. □

The next step is to define the appropriate covering relation $\sqsubseteq_{\mathsf{gl}}$ that is a simulation w.r.t. $\rightarrow_{\mathsf{gl}}$. One option would be to just require equality on the third component of triples. The definition we adopt allows for more coverings.

Definition 8.7 (Node-covering relation $\sqsubseteq_{\mathsf{gl}}$)**.** For nodes of a POR-LZG we define a covering relation: $(q, \mathsf{Z}, r) \sqsubseteq_{\mathsf{gl}} (q', \mathsf{Z}', r')$ if $q = q'$, $\mathsf{Z} \sqsubseteq^D_M \mathsf{Z}'$ and $r' \leq r$.

Lemma 8.13. *If $\mathcal{N}$ is a global-local system, then $\sqsubseteq_{\mathsf{gl}}$ is a simulation relation in $\mathsf{POR\text{-}LZG}_{\mathsf{gl}}(\mathcal{N})$.*

Proof. Let $(q, \mathsf{Z}, r) \sqsubseteq_{\mathsf{gl}} (q, \mathsf{Z}', r')$ in $\mathsf{POR\text{-}LZG}_{\mathsf{gl}}(\mathcal{N})$. Observe that by the definition of the covering: the first components must be the same, $\mathsf{Z} \sqsubseteq^D_M \mathsf{Z}'$, and $r' \leq r$. We show that if $(q, \mathsf{Z}, r) \xrightarrow{a}_{\mathsf{gl}} (q_1, \mathsf{Z}_1, r_1)$, then there exists a node $(q_1, \mathsf{Z}'_1, r_1)$ in $\mathsf{POR\text{-}LZG}_{\mathsf{gl}}(\mathcal{N})$ such that $(q, \mathsf{Z}', r') \xrightarrow{a}_{\mathsf{gl}} (q_1, \mathsf{Z}'_1, r_1)$, and $(q_1, \mathsf{Z}_1, r_1) \sqsubseteq_{\mathsf{gl}} (q_1, \mathsf{Z}'_1, r_1)$. Observe that the simulating node $(q_1, \mathsf{Z}'_1, r_1)$ differs from (q_1, Z_1, r_1) only in the second component.

Since $\mathsf{Z} \sqsubseteq^D_M \mathsf{Z}'$, we know by Lemma 6.10 that $(q, \mathsf{Z}') \xrightarrow{a} (q_1, \mathsf{Z}'_1)$ for some Z'_1 with $\mathsf{Z}_1 \sqsubseteq^D_M \mathsf{Z}'_1$.

If a is a global action, we have $(q, \mathsf{Z}', r') \xrightarrow{a}_{\mathsf{gl}} (q_1, \mathsf{Z}'_1, 0)$ in $\mathsf{POR\text{-}LZG}_{\mathsf{gl}}(\mathcal{N})$. As in this case also $r_1 = 0$, we obtain the following desired result: $(q_1, \mathsf{Z}_1, r_1) \sqsubseteq_{\mathsf{gl}} (q_1, \mathsf{Z}'_1, r'_1)$.

Otherwise, a is a local action of some process A_{r_1}. Since a is enabled from (q, Z, r) in $\mathsf{POR\text{-}LZG}_{\mathsf{gl}}(\mathcal{N})$, we know that $r_1 \geq r$. As $r' \leq r$, from the definition of $\rightarrow_{\mathsf{gl}}$, we know that $(q', \mathsf{Z}', r') \xrightarrow{a}_{\mathsf{gl}} (q_1, \mathsf{Z}'_1, r_1)$. It is clear that $(q_1, \mathsf{Z}_1, r_1) \sqsubseteq_{\mathsf{gl}} (q_1, \mathsf{Z}'_1, r_1)$. □

Thanks to Lemma 8.12 and Lemma 8.13, we can use Theorem 8.2 to get a reachability algorithm for global-local systems.

Lemma 8.14. *Algorithm 5 using $\mathsf{Next}_{\mathsf{gl}}$ and $\sqsubseteq_{\mathsf{gl}}$ solves the reachability problem for global-local systems.*

8.3.2 Extended global-local systems

In order to broaden the applicability of our global-local POR technique, we now propose a more general class of systems, which we call *extended global-local systems*, and show how to apply global-local POR on these systems. In these extended global-local systems, we do not impose restrictions on communications between processes. The price to pay will be reduced concurrency, or even no concurrency at all if there is a communication action between every pair of processes.

As before, an action is *global* if it synchronizes all the processes of the network. Otherwise an action is *non-global*. As a consequence, a local action of a process is a non-global action, but there are also communication actions that are non-global. A network $\mathcal{N} = \{A_1, A_2, \cdots, A_n\}$ is said to be an extended global-local system if it satisfies property (gl-final) from page 167. Thus, given an accepting state $(q_1, \cdots, q_n)$ of $\mathcal{N}$ where q_i is a state of A_i for $1 \leq i \leq n$, each incoming transition to q_i is labelled by a global action. Recall that by a remark on page 167, every network can be transformed to a network satisfying (gl-final).

Successor computation for extended global-local systems

The idea is to group processes into equivalence classes: two processes belong to the same equivalence class if there is a non-global action that can synchronize them. The approach we are going to propose can be viewed as applying the standard global-local POR algorithm to a system where processes are "total synchronizations of equivalence classes of processes".

We describe in more detail the equivalence relation on processes. We say that two processes *can communicate with each other* if there is a non-global action with the two processes in its domain. We will work with equivalence classes of "can communicate with each other" relation. To have some notation for these equivalence classes, we define a mapping $\mathsf{groupid}$ that assigns an integral value to each process A_i: it is the smallest value j such that process A_j belongs to the equivalence class of A_i. Notice that if two processes can synchronize on a non-global action, then they have the same $\mathsf{groupid}$. We define $\mathsf{groupid}$ of a non-global action to be the $\mathsf{groupid}$ of processes involved in this action. For convenience, we also define $\mathsf{groupid}(a) = 0$ for all global actions a.

We can now define a restricted transition relation $\rightarrow_{\mathsf{egl}}$. It is a generalization of Definition 8.6.

Definition 8.8. We define a subset $\rightarrow_{\mathsf{egl}}$ of $\rightarrow$ transitions. Given a node (q, Z, r) of $\mathsf{POR\text{-}LZG}$ and a transition $(q, \mathsf{Z}) \xrightarrow{a} (q', \mathsf{Z}')$ of LZG, we have $(q, \mathsf{Z}, r) \xrightarrow{a}_{\mathsf{egl}} (q', \mathsf{Z}', \mathsf{groupid}(a))$ provided $\mathsf{groupid}(a) \geq r$, or a is a global action.

An algorithm for computing $\rightarrow_{\mathsf{egl}}$ transitions in presented in Algorithm 7.

Algorithm 7 $\mathsf{Next}_{\mathsf{egl}}(q, \mathsf{Z}, r)$

Input: A node (q, Z, r) of the POR-zone graph of an extended global-local network of timed automata A.
Output: $\mathsf{Source}(q, \mathsf{Z}, r)$

```
1: Source := ∅
2: for every transition (q, Z) -a-> (q', Z') do
3:     if a is global then
4:         Add (q', Z', 0) to Source
5:     else
6:         r' = groupid(a)
7:         if (r' ≥ r) then
8:             Add (q', Z', r') to Source
9: return Source
```

Lemma 8.15. *The successor relation $\rightarrow_{\mathsf{egl}}$ is complete for reachability for transition systems of extended global-local systems.*

Proof. The proof follows closely along the lines of the completeness for global-local systems (Lemma 8.12.)

Let $\mathsf{POR\text{-}LZG}_{\mathsf{egl}}(\mathcal{N})$ be the reduced transition system of $\mathsf{POR\text{-}LZG}(\mathcal{N})$ induced by $\rightarrow_{\mathsf{egl}}$. We show that the reduced system is reachability complete. Consider a path:

$$(q_0, \mathsf{Z}_0, 0) \xrightarrow{\sigma_1} (q_1, \mathsf{Z}_1, r_1) \xrightarrow{g_1} (q_2, \mathsf{Z}_2, 0) \xrightarrow{\sigma_2} (q_3, \mathsf{Z}_3, r_3) \xrightarrow{g_2} (q_4, \mathsf{Z}_4, 0) \cdots$$
$$(q_{2n-2}, \mathsf{Z}_{2n-2}, 0) \xrightarrow{\sigma_n} (q_{2n-1}, \mathsf{Z}_{2n-1}, r_{2n-1}) \xrightarrow{g_n} (q_{2n}, \mathsf{Z}_{2n}, 0)$$

in $\mathsf{POR\text{-}LZG}(\mathcal{N})$, where σ_i's are sequences of non-global actions and g_i's are global actions. Observe that after the execution of a global action, the rank of a configuration is 0.

It is enough to show that there are $r'_1, \ldots, r'_{2n-1}$ and $\sigma'_1, \ldots, \sigma'_n$ such that a path of the form

$$(q_0, \mathsf{Z}_0, 0) \xrightarrow{\sigma'_1}_{\mathsf{egl}} (q_1, \mathsf{Z}_1, r'_1) \xrightarrow{g_1}_{\mathsf{egl}} (q_2, \mathsf{Z}_2, 0) \xrightarrow{\sigma'_2}_{\mathsf{egl}} (q_3, \mathsf{Z}_3, r'_3) \xrightarrow{g_2}_{\mathsf{egl}} (q_4, \mathsf{Z}_4, 0) \cdots$$
$$(q_{2n-2}, \mathsf{Z}_{2n-2}, 0) \xrightarrow{\sigma'_n}_{\mathsf{egl}} (q_{2n-1}, \mathsf{Z}_{2n-1}, r'_{2n-1}) \xrightarrow{g_n}_{\mathsf{egl}} (q_{2n}, \mathsf{Z}_{2n}, 0)$$

exists in $\mathsf{POR\text{-}LZG}_{\mathsf{egl}}(\mathcal{N})$. We will ensure that $\sigma'_i \sim \sigma_i$.

First, we know that the initial node of $\mathsf{POR\text{-}LZG}(\mathcal{N})$ has rank 0. Further, recall that, if a global action is feasible from a node of the $\mathsf{POR\text{-}LZG}(\mathcal{N})$, then it is allowed by $\rightarrow_{\mathsf{egl}}$ (irrespective of the rank of the node).

Next, consider a segment consisting of non-global actions in the aforementioned path in $\mathsf{POR\text{-}LZG}(\mathcal{N})$:

$$(q_{i_0}, \mathsf{Z}_{i_0}, r_{i_0}) \xrightarrow{a_1} (q_{i_1}, \mathsf{Z}_{i_1}, r_{i_1}) \xrightarrow{a_2} (q_{i_2}, \mathsf{Z}_{i_2}, r_{i_2}) \cdots \xrightarrow{a_l} (q_{i_l}, \mathsf{Z}_{i_l}, r_{i_l}).$$

We know that two actions with different $\mathsf{groupid}$ have disjoint domains, and hence, can be commuted thanks to Lemma 8.6. For every j, consider the subsequence σ^j of the above sequence consisting only of actions with $\mathsf{groupid} = j$. Since there may be no actions of some process in a segment, we consider only non-empty sequences: $\sigma^{j_1}, \sigma^{j_2}, \ldots, \sigma^{j_k}$ for some $1 \leq j_1 < j_2 < \cdots < j_k \leq n$. Recalling the definition of $\rightarrow_{\mathsf{egl}}$ we see that in $\mathsf{POR\text{-}LZG}_{\mathsf{egl}}(\mathcal{N})$ we have a path

$$(q_{i_0}, \mathsf{Z}_{i_0}, 0) \xrightarrow{\sigma^{j_1}}_{\mathsf{egl}} (q'_{j_1}, \mathsf{Z}'_{j_1}, j_1) \xrightarrow{\sigma^{j_2}}_{\mathsf{egl}} (q'_{j_2}, \mathsf{Z}'_{j_2}, j_2) \cdots \xrightarrow{\sigma^{j_k}}_{\mathsf{egl}} (q_{j_k}, \mathsf{Z}_{j_k}, j_k) \ .$$

This shows how, for all i, we can obtain σ'_i from σ_i, and this completes the proof. □

To complete the construction we need to define a covering relation and show that it is a simulation w.r.t. $\rightarrow_{\mathsf{egl}}$. We show that we can in fact, use the same relation $\sqsubseteq_{\mathsf{gl}}$ as in Definition 8.7.

Lemma 8.16. *If $\mathcal{N}$ is an extended global-local system, then $\sqsubseteq_{\mathsf{gl}}$ is a simulation relation in $\mathsf{POR\text{-}LZG}_{\mathsf{egl}}(\mathcal{N})$.*

Proof. The proof is very similar to that of Lemma 8.13.

Let $(q, \mathsf{Z}, r) \sqsubseteq_{\mathsf{gl}} (q, \mathsf{Z}', r')$ in $\mathsf{POR\text{-}LZG}_{\mathsf{egl}}(\mathcal{N})$. Observe that by the definition of the covering: the first components must be the same, $\mathsf{Z} \sqsubseteq_M^D \mathsf{Z}'$, and $r' \leq r$. We show that if $(q, \mathsf{Z}, r) \xrightarrow{a}_{\mathsf{egl}} (q_1, \mathsf{Z}_1, r_1)$, then there exists a node $(q_1, \mathsf{Z}'_1, r_1)$ in $\mathsf{POR\text{-}LZG}_{\mathsf{egl}}(\mathcal{N})$ such that $(q, \mathsf{Z}', r') \xrightarrow{a}_{\mathsf{egl}} (q_1, \mathsf{Z}'_1, r_1)$, and $(q_1, \mathsf{Z}_1, r_1) \sqsubseteq_{\mathsf{gl}} (q_1, \mathsf{Z}'_1, r_1)$. Observe that the simulating node $(q_1, \mathsf{Z}'_1, r_1)$ differs only on the second component from (q_1, Z_1, r_1).

Since $\mathsf{Z} \sqsubseteq_M^D \mathsf{Z}'$, we know by Lemma 6.10 that $(q, \mathsf{Z}') \xrightarrow{a} (q_1, \mathsf{Z}'_1)$ for some Z'_1 with $\mathsf{Z}_1 \sqsubseteq_M^D \mathsf{Z}'_1$.

If a is a global action, we have $(q, \mathsf{Z}', r') \xrightarrow{a}_{\mathsf{egl}} (q_1, \mathsf{Z}'_1, 0)$ in $\mathsf{POR\text{-}LZG}_{\mathsf{egl}}(\mathcal{N})$. As in this case also $r_1 = 0$, we have $(q_1, \mathsf{Z}_1, r_1) \sqsubseteq_{\mathsf{gl}} (q_1, \mathsf{Z}'_1, r'_1)$.

Otherwise, a is a non-global action and $\mathsf{groupid}(a) = r_1$. Since a is enabled from (q, Z, r) in $\mathsf{POR\text{-}LZG}_{\mathsf{egl}}(\mathcal{N})$, we know that $r_1 \geq r$. As $r' \leq r$, from the definition of $\rightarrow_{\mathsf{egl}}$, we know that a is enabled from (q', Z', r') and we have $(q', \mathsf{Z}', r') \xrightarrow{a}_{\mathsf{egl}} (q_1, \mathsf{Z}'_1, r_1)$. Here, it is clear that $(q_1, \mathsf{Z}_1, r_1) \sqsubseteq_{\mathsf{gl}} (q_1, \mathsf{Z}'_1, r_1)$. □

Thanks to Lemma 8.15 and Lemma 8.16, we can use Theorem 8.2 to get a reachability algorithm for extended global-local systems.

Lemma 8.17. *Algorithm 5 using $\mathsf{Next}_{\mathsf{egl}}$ and $\sqsubseteq_{\mathsf{gl}}$ solves the reachability problem for extended global-local systems.*

8.4. Client-server POR

In this section, we propose a POR implementation that is customized and targeted at a class of networks of timed automata called *client-server systems*. We describe a procedure $\mathsf{Next}_{\mathsf{cs}}$ that computes the successors of a node of the POR-zone graph and a covering relation $\sqsubseteq_{\mathsf{cs}}$ for the nodes of the POR-zone graph of a client-server system.

8.4.1 Client-server systems

We first briefly recall the properties of client-server systems discussed in detail in Section 7.3. In these networks, we have a server process (denoted by S) and several client processes (denoted by $C_1, C_2, \cdots, C_k$). For convenience, we suppose that S, is the process number 0. These systems can have three kinds of actions

- local actions in a client, whose domain is $\{C_i\}$ for some i,
- local actions in the server which has domain $\{S\}$,
- synchronization actions involving a client and the server, that have domain $\{S, C_i\}$ for some i.

So, for every action a, $\mathsf{dom}(a)$ is either a singleton $\{S\}$ or $\{C_i\}$, or a pair $\{S, C_i\}$, for some client C_i. We refer to synchronizations involving processes S and C_i as *communication* actions.

Further, the definition of client-server systems also imposes conditions on how an accepting state can be reached in the system - in a client-server system, each process needs to execute a communication action with the server just before reaching an accepting state.

Recall that an accepting state is a global state, i.e., a tuple $(q_0, q_1, \cdots, q_n)$, where q_0 is a state of S and q_i is a state of process in C_i, for $1 \leq i \leq n$. Then, the aforementioned implies that given an accepting state $(q_0, q_1, \cdots, q_n)$ of $\mathcal{N}$, for $1 \leq i \leq n$, each incoming transition to q_i is labelled by a communication action with the server.

Remark. *Suppose that $\mathcal{N}$ is a network which satisfies all the properties of client-server systems except the condition on the reachability of accepting states. $\mathcal{N}$ can be transformed to a client-server system by introducing a dummy accepting state in each process, and adding incoming transitions from all the (original) accepting states to this dummy accepting state, such that each of these transitions is labelled by a new communication action with the server.*

Successor computation for client-server systems

The idea of $\rightarrow_{\mathsf{cs}}$ is to choose one client and continue executing actions of that client till it does a communication action; then, a next client is chosen. The third component r, keeps the index of the process that has executed the last action.

Definition 8.9. We define a subset $\rightarrow_{\mathsf{cs}}$ of $\rightarrow$ transitions of POR-LZG. Given a node (q, Z, r) of POR-LZG and a transition $(q, \mathsf{Z}) \xrightarrow{a} (q', \mathsf{Z}')$ of LZG, we have $(q, \mathsf{Z}, r) \xrightarrow{a}_{\mathsf{cs}} (q', \mathsf{Z}', r')$ if either:

- when $r = 0$: a is a local action of the client r', or a involves the server and $r' = 0$.
- when $r > 0$: a is a local action of the client r and $r' = r$, or a is a communication action of the client r with the server and $r' = 0$.

The associated function $\mathsf{Next}_{\mathsf{cs}}$ computing the set of $\rightarrow_{\mathsf{cs}}$ successors of a node is presented in Algorithm 8.

Algorithm 8 $\mathsf{Next}_{\mathsf{cs}}(q, \mathsf{Z}, r)$

Input: A node (q, Z, r) of the POR-zone graph of a network of timed automata $\mathcal{N} = \langle S, C_1, C_2, \cdots C_k \rangle$
Output: $\mathsf{Succ}(q, \mathsf{Z}, r)$

```
1: Succ := ∅
2: for every transition (q, Z) -a-> (q', Z') do
3:     if r = 0 or C_r ∈ dom(a) then
4:         if S ∈ dom(a) then
5:             r' = 0
6:         else
7:             r' = i for some C_i ∈ dom(a)
8:         Add (q', Z', r') to Succ
9: return Succ
```

Lemma 8.18. *The successor relation $\rightarrow_{\mathsf{cs}}$ is complete for reachability for transitions systems of client-server systems.*

Proof. Let $\mathsf{POR\text{-}LZG}_{\mathsf{cs}}(\mathcal{N})$ be the reduced transition system of $\mathsf{POR\text{-}LZG}(\mathcal{N})$ induced by $\rightarrow_{\mathsf{cs}}$. We show that the reduced transition system is reachability complete. The proof is by induction on the length of a path to a final state.

Consider a path

$$(q_0, \mathsf{Z}_0, 0) \xrightarrow{\sigma} (q_1, \mathsf{Z}_1, r_1) \xrightarrow{b} (q_2, \mathsf{Z}_2, 0) \xrightarrow{\rho} (q_3, \mathsf{Z}_3, 0)$$

in $\mathsf{POR\text{-}LZG}(\mathcal{N})$, where b is the first action of the server in the sequence. Hence, all actions in σ are local actions of clients. Observe that such an

action b must exist as, by our assumption on client-server systems, each process needs to execute a communication action just before reaching an accepting state. Moreover this implies that the rank in the last configuration is 0, but there may be other ranks on ρ in the segment of the path between q_2 and q_3.

If b is a local action of the server, all actions in σ commute with b. By Lemma 8.6, the following is also a path in $\mathsf{POR\text{-}LZG}(\mathcal{N})$:

$$(q_0, \mathsf{Z}_0, 0) \xrightarrow{b} (q_1', \mathsf{Z}_1', 0) \xrightarrow{\sigma} (q_2', \mathsf{Z}_2', r_2') \xrightarrow{\rho} (q_3, \mathsf{Z}_3, 0)$$

By the induction assumption, we have a path $(q_1', \mathsf{Z}_1', 0) \xrightarrow{\sigma\rho}_{\mathsf{cs}} (q_3, \mathsf{Z}_3, 0)$. We also have $(q_0, \mathsf{Z}_0, 0) \xrightarrow{b}_{\mathsf{cs}} (q_1', \mathsf{Z}_1', 0)$ by the definition of $\to_{\mathsf{cs}}$. This gives us a path in $\mathsf{POR\text{-}LZG_{cs}}(\mathcal{N})$.

The other case is when b is a communication action with a client r. In this case, we split σ into two subsequences: σ_1 consisting of local actions of client r, and σ_2 consisting of local actions of other clients. All actions of σ_2 commute with b. By Lemma 8.6 the following is also a path in $\mathsf{POR\text{-}LZG}(\mathcal{N})$:

$$(q_0, \mathsf{Z}_0, 0) \xrightarrow{\sigma_1} (q_1', \mathsf{Z}_1', 0) \xrightarrow{b} (q_1'', \mathsf{Z}_1'', 0) \xrightarrow{\sigma_2} (q_2', \mathsf{Z}_2', r_2') \xrightarrow{\rho} (q_3, \mathsf{Z}_3, 0)$$

By the induction assumption, we have $(q_1'', \mathsf{Z}_1'', 0) \xrightarrow{\sigma_2\rho}_{\mathsf{cs}} (q_3, \mathsf{Z}_3, 0)$ in $\mathsf{POR\text{-}LZG_{cs}}(\mathcal{N})$. From the definition of $\to_{\mathsf{cs}}$ we get

$$(q_0, \mathsf{Z}_0, 0) \xrightarrow{\sigma_1}_{\mathsf{cs}} (q_1', \mathsf{Z}_1', r) \xrightarrow{b}_{\mathsf{cs}} (q_1'', \mathsf{Z}_1'', 0)$$

This gives us a desired execution. □

In order to apply our reachability testing algorithm, it remains to define a covering relation $\sqsubseteq_{\mathsf{cs}}$ that is a simulation w.r.t. $\to_{\mathsf{cs}}$.

Definition 8.10. For nodes of $\mathsf{POR\text{-}LZG}(\mathcal{N})$, we define a covering relation $(q, \mathsf{Z}, r) \sqsubseteq_{\mathsf{cs}} (q', \mathsf{Z}', r')$ if $q = q'$, $\mathsf{Z} \sqsubseteq_M^D \mathsf{Z}'$ and $r' = r$ or $r' = 0$.

Lemma 8.19. *If $\mathcal{N}$ is a client-server system, then $\sqsubseteq_{\mathsf{cs}}$ is a simulation relation in $\mathsf{POR\text{-}LZG_{cs}}(\mathcal{N})$.*

Proof. Let $(q, \mathsf{Z}, r) \sqsubseteq_{\mathsf{cs}} (q, \mathsf{Z}', r')$ in $\mathsf{POR\text{-}LZG_{cs}}(\mathcal{N})$. Observe that by the definition of the covering: the first components must be the same, $\mathsf{Z} \sqsubseteq_M^D \mathsf{Z}'$, and $r = r'$, or $r' = 0$. We show that if $(q, \mathsf{Z}, r) \xrightarrow{a}_{\mathsf{cs}} (q_1, \mathsf{Z}_1, r_1)$, then there exists a node $(q_1, \mathsf{Z}_1', r_1)$ in $\mathsf{POR\text{-}LZG_{cs}}(\mathcal{N})$ such that $(q, \mathsf{Z}', r') \xrightarrow{a}_{\mathsf{cs}} (q_1, \mathsf{Z}_1', r_1)$, and $(q_1, \mathsf{Z}_1, r_1) \sqsubseteq_{\mathsf{cs}} (q_1, \mathsf{Z}_1', r_1)$. Observe that the simulating node $(q_1, \mathsf{Z}_1', r_1)$ differs from (q_1, Z_1, r_1) only in the second component.

Since $\mathsf{Z} \sqsubseteq_M^D \mathsf{Z}'$, we know by Lemma 6.10 that $(q, \mathsf{Z}') \xrightarrow{a} (q_1, \mathsf{Z}_1')$ for some Z_1' with $\mathsf{Z}_1 \sqsubseteq_M^D \mathsf{Z}_1'$.

Both when $r = r'$ or $r' = 0$, we have $(q, \mathsf{Z}', r') \xrightarrow{a}_{\mathsf{cs}} (q_1, \mathsf{Z}_1', r_1)$ in $\mathsf{POR\text{-}LZG_{cs}}(\mathcal{N})$. As a consequence, we have $(q_1, \mathsf{Z}_1, r_1) \sqsubseteq_{\mathsf{cs}} (q_1, \mathsf{Z}_1', r_1)$. □

Thanks to Lemma 8.18 and Lemma 8.19, we can use Theorem 8.2 to get a reachability algorithm for client-server systems.

Lemma 8.20. *Algorithm 5 using* $\mathsf{Next}_{\mathsf{cs}}$ *and* $\sqsubseteq_{\mathsf{cs}}$ *solves the reachability problem for client-server systems.*

8.4.2 Extended client-server systems

In this section, we present the application of the client-server POR technique to a more general class of systems, referred to as *extended client-server systems*. We hope that this extension would help broaden the scope of our client-server POR technique.

In these systems, we allow arbitrary synchronizations involving non-server processes. Thus, we have a server process and disjoint groups of client processes that have synchronizations within the groups. Our approach can be viewed as considering each such group of client processes as a client and applying client-server POR on the resultant client-server system.

We use an idea similar to the one we used for extended global-local systems (see Section 8.3.2): we group client processes into equivalence classes. We say that two client processes *can communicate with each other* if there is an action with the two processes in its domain. This relation "can communicate with each other" is an equivalence relation and it partitions the set of client processes into equivalence classes. The equivalence class of a client process C is the set of client processes C' such that C and C' can communicate with each other without the server, i.e., there is sequence of communication actions not involving the server linking C and C'.

Given a network $\{C_0, C_1, C_2, \cdots, C_k\}$, we assume that the process C_0 is the server, while the others are clients. We define a mapping $\mathsf{groupid}$ that assigns an integral value to each process C_i such that

- The server has $\mathsf{groupid} = 0$. It is the unique process with $\mathsf{groupid} = 0$.
- For a client C_i, we define $\mathsf{groupid}(C_i)$ as the smallest $j > 0$ such that the process C_j belongs to the equivalence class of C_i.

It follows from the definition of $\mathsf{groupid}$, that if there is an action synchronizing two clients C_i and C_j such that $i, j > 0$, then $\mathsf{groupid}(C_i) = \mathsf{groupid}(C_j)$.

The function $\mathsf{groupid}$ defines an equivalence relation on clients. We have three categories of actions:

- *client actions* involving only clients with the same $\mathsf{groupid}$,
- *local server actions* involving only the server,
- *communication actions* involving the server and some clients, all with the same $\mathsf{groupid}$.

We define the groupid of a client action as the groupid of processes involved in this action. Otherwise, groupid of an action is 0.

The degenerate case of this situation is when all the clients have the same groupid. In this case there is no restriction on possible communication actions, but there will also be no gain when applying the method we are going to present.

The only restriction we impose is that the final state of each process should be only reachable by a communication action with the server.

Successor computation for extended client-server systems

The idea of $\rightarrow_{\mathsf{ecs}}$ is to choose one groupid and let the processes of this groupid execute till a communication action with the server; after that some other group of processes is chosen. The third component now keeps track of the groupid of the last action.

Definition 8.11. We define a subset $\rightarrow_{\mathsf{ecs}}$ of $\rightarrow$ transitions. Given a node (q, Z, r) of POR-LZG and a transition $(q, \mathsf{Z}) \xrightarrow{a} (q', \mathsf{Z}')$ of LZG we have $(q, \mathsf{Z}, r) \xrightarrow{a}_{\mathsf{ecs}} (q', \mathsf{Z}', \mathsf{groupid}(a))$ under the condition that either: (i) $r = 0$, or (ii) a is an action involving processes with $\mathsf{groupid} = r$.

Algorithm 9 $\mathsf{Next}_{\mathsf{ecs}}(q, \mathsf{Z}, r)$

Input: A node (q, Z, r) of the POR-zone graph of a network of timed automata $\mathcal{N}$ and a map $\mathsf{groupid} : \{S, C_1, C_2, \cdots C_k\} \mapsto \mathbb{N}$
Output: $\mathsf{Source}(q, \mathsf{Z}, r)$

```
1: Source := ∅
2: for every transition (q, Z) -a-> (q', Z') do
3:     if r = 0 or there is C_i ∈ dom(a) with groupid(C_i) = r then
4:         if S ∈ dom(a) then
5:             r' = 0
6:         else
7:             r' = groupid(a)
8:         Add (q', Z', r') to Source
9: return Source
```

Lemma 8.21. *The successor relation $\rightarrow_{\mathsf{ecs}}$ is complete for reachability for transition systems of extended client-server systems.*

Proof. Let $\mathsf{POR\text{-}LZG}_{\mathsf{ecs}}(\mathcal{N})$ be the reduced transition system of $\mathsf{POR\text{-}LZG}(\mathcal{N})$ induced by $\rightarrow_{\mathsf{ecs}}$. We show that the reduced system is reachability complete. The proof is by induction on the length of a path to a final state.

Consider a path

$$(q_0, \mathsf{Z}_0, 0) \xrightarrow{\sigma} (q_1, \mathsf{Z}_1, r_1) \xrightarrow{b} (q_2, \mathsf{Z}_2, r_2) \xrightarrow{\rho} (q_3, \mathsf{Z}_3, r_3)$$

in POR-LZG($\mathcal{N}$), where b is the first action of the server in the sequence. Such an action b must exist as, by our assumption on extended client-server systems, the final state can be reached only with a communication action. Hence, all actions in σ are actions not involving the server.

If b is a local action of the server, all actions in σ commute with b. By Lemma 8.6 the following is also a path in POR-LZG($\mathcal{N}$):

$$(q_0, \mathsf{Z}_0, 0) \xrightarrow{b} (q_1', \mathsf{Z}_1', 0) \xrightarrow{\sigma} (q_2', \mathsf{Z}_2', r_2') \xrightarrow{\rho} (q_3, \mathsf{Z}_3, r_3)$$

By the induction assumption we have a path $(q_1', \mathsf{Z}_1', 0) \xrightarrow{\sigma\rho}_{\mathsf{cs}} (q_3, \mathsf{Z}_3, r_3)$. We have also $(q_0, \mathsf{Z}_0, 0) \xrightarrow{b}_{\mathsf{cs}} (q_1', \mathsf{Z}_1', 0)$ by the definition of $\to_{\mathsf{cs}}$. This gives us a path in POR-LZG$_{\mathsf{ecs}}(\mathcal{N})$.

The other case is when b is a communication action with clients of groupid $= r$. In this case, we split σ into two subsequences: σ_1 consisting of actions of client with groupid $= r$, and σ_2 consisting of actions of other clients. All actions of σ_2 commute with b. By Lemma 8.6 the following is also a path in POR-LZG($\mathcal{N}$):

$$(q_0, \mathsf{Z}_0, 0) \xrightarrow{\sigma_1} (q_1', \mathsf{Z}_1', 0) \xrightarrow{b} (q_1'', \mathsf{Z}_1'', 0) \xrightarrow{\sigma_2} (q_2', \mathsf{Z}_2', r_2') \xrightarrow{\rho} (q_3, \mathsf{Z}_3, r_3)$$

By the induction assumption we have $(q_1'', \mathsf{Z}_1'', 0) \xrightarrow{\sigma_2\rho}_{\mathsf{cs}} (q_3, \mathsf{Z}_3, r_3)$ in POR-LZG$_{\mathsf{ecs}}(\mathcal{N})$. From the definition of $\to_{\mathsf{cs}}$ we get

$$(q_0, \mathsf{Z}_0, 0) \xrightarrow{\sigma_1}_{\mathsf{cs}} (q_1', \mathsf{Z}_1', r) \xrightarrow{b}_{\mathsf{cs}} (q_1'', \mathsf{Z}_1'', 0)$$

This gives us an execution of the desired form. □

For the node covering relation, we use the same relation $\sqsubseteq_{\mathsf{cs}}$ as for client-server systems (Definition 8.10). The proof of the lemma is practically the same as that of Lemma 8.19.

Lemma 8.22. *$\sqsubseteq$ is a simulation relation for* POR-LZG($\mathcal{N}$)*, if $\mathcal{N}$ is an extended client-server system.*

Thanks to Lemma 8.21 and Lemma 8.22, we can use Theorem 8.2 to get a reachability algorithm for extended client-server systems.

Lemma 8.23. *Algorithm 5 using* Next$_{\mathsf{ecs}}$ *and* $\sqsubseteq_{\mathsf{cs}}$ *solves the reachability problem for extended client-server systems.*

8.5. Optimizations for POR reachability algorithm

In this section, we study some properties of global-local and client-server systems and use these properties to propose some optimizations for the respective POR implementations, namely global-local POR and client-server

POR. At the end of the section, we present the revised versions of the Next procedures with these optimizations incorporated - $\mathsf{Next}^*_{\mathsf{gl}}$ (Algorithm 10) and $\mathsf{Next}^*_{\mathsf{cs}}$ (Algorithm 11), respectively. Note that, in this section, we work with the more generic framework of extended global-local systems and extended client server systems.

Global-local POR

Recall that the conditions on global-local systems stipulate that an accepting state in an extended global-local system can only be reached by executing a global action at the end. Here, we discuss some optimizations to the $\mathsf{Next}_{\mathsf{egl}}$ procedure that exploit this property of extended global-local systems. Each of these optimizations essentially identifies nodes of the $\mathsf{POR\text{-}LZG}$ of the network from which paths cannot contain global actions, and stops the exploration from these nodes.

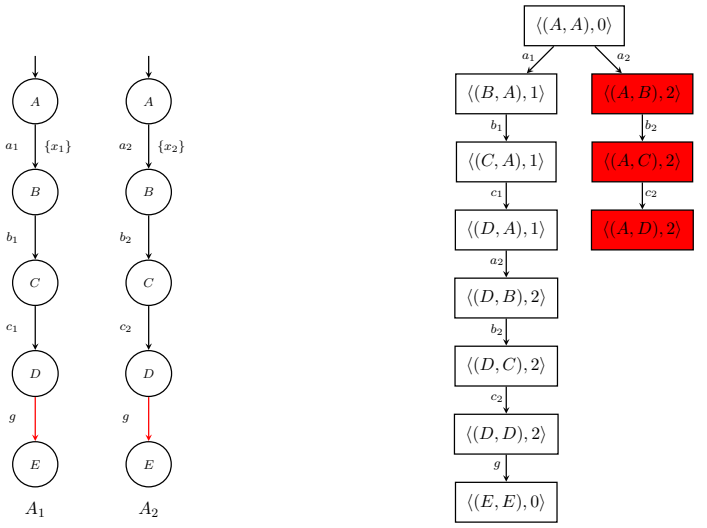
First optimization For the first optimization, we introduce the notion of $\mathsf{gl}(i, q, r)$, which is an over-approximation of the set of global actions that process A_i can execute from a node with state q and rank r. Two things are to be noted here. First, if $\mathsf{groupid}(A_i) < r$, then from a node with rank r, process A_i is not allowed to execute its local actions before the next global action. Second, this is a syntactic construct – the timing information associated to a node has no bearing on the value of $\mathsf{gl}(i, q, r)$. We now make this notion precise.

Definition 8.12 ($\mathsf{gl}(i, q, r)$)**.** Let $\mathcal{N} = \langle A_1, A_2, \cdots A_k \rangle$ be a global-local system. We define a set of global actions $\mathsf{gl}(i, q, r)$ for $i, r \in \{1, \ldots, k\}$, and q a state of $\mathcal{N}$. If $\mathsf{groupid}(A_i) < r$, then $\mathsf{gl}(i, q, r)$ is the set of global actions g such that there is an outgoing transition labelled g from q. If $\mathsf{groupid}(A_i) \geq r$, then $\mathsf{gl}(i, q, r)$ is the set of global actions g such that there is a state q' reachable from q by a sequence of non-global transitions whose $\mathsf{groupid}$ is $\mathsf{groupid}(A_i)$ and an outgoing transition labelled g from q'.

We use this notion to curtail exploration from those nodes of $\mathsf{POR\text{-}LZG}(\mathcal{N})$ from which we are sure that there is no run containing a global action.

Lemma 8.24. *Let $\mathcal{N}$ be an extended global-local system, and let (q, Z, r) be a node of $\mathsf{POR\text{-}LZG}_{\mathsf{egl}}(\mathcal{N})$. If there exist two processes A_i and A_j such that $\mathsf{gl}(i, q, r) \cap \mathsf{gl}(j, q, r) = \emptyset$, then an accepting state cannot be reached from (q, Z, r).*

Proof. Recall that, by the definition of global-local systems, an accepting path should end with a global action. If $\mathsf{gl}(i, q, r) \cap \mathsf{gl}(j, q, r) = \emptyset$, then this means that there is no global action that can be executed from (q, Z, r) or any node reachable from it (independently of Z). □



(a) Network $\mathcal{N}$ (b) POR-zone graph of $\mathcal{N}$

Figure 8.1: An example that illustrates the effect of Optimization 1 of global-local POR

As a corollary, we get the following optimization.

Optimization 1. For an extended global-local system $\mathcal{N}$, if (q, Z, r) is a node of $\mathsf{POR\text{-}LZG}_{\mathsf{egl}}(\mathcal{N})$ such that $\mathsf{gl}(i, q, r) \cap \mathsf{gl}(j, q, r) = \emptyset$, for some i, j, then (q, Z, r) need not be stored and its successors need not be explored.

Example for Optimization 1: Consider the network $\mathcal{N}$ presented in Figure 8.1a. We present the POR-zone graph, $\mathsf{POR\text{-}LZG}_{\mathsf{gl}}(\mathcal{N})$, as produced by Algorithm 6 in Figure 8.1b. The nodes that will be removed by Optimization 1 are highlighted in red. To understand the working of this optimization, we pick one such node, $\langle (A, B), 2\rangle$, and explain why this node is removed by Optimization 1. Observe that the location A of process A_1 does not have any outgoing global actions. This means that $\mathsf{gl}(1, B, 2) = \emptyset$ and as a consequence, $\mathsf{gl}(1, B, 2) \cap \mathsf{gl}(2, B, 2) = \emptyset$. Hence, the node $\langle (A, B), 2\rangle$ can be removed, and its successors need not be explored.

Second optimization Here, we present another optimization, Optimization 2 for the Next procedure for global-local systems. Before presenting the optimization, we first state Lemma 8.25 that gives a key property of the transition relation of the reduced POR-zone graph. Note that we refer to the set of clocks $X_i \cup t_i$ as clocks of process A_i. Then, $\mathsf{Z}|_{<p}$ denotes the projection of Z to the clocks of processes $A_1, A_2, \cdots, A_{p-1}$.

Lemma 8.25. *Let $\mathcal{N}$ be an extended global-local system. Let (q, Z, r) be a node of $\mathsf{POR\text{-}LZG}_{\mathsf{egl}}(\mathcal{N})$ and let $(q, \mathsf{Z}, r) \xrightarrow{a}_{\mathsf{egl}} (q', \mathsf{Z}', r')$ be a transition in $\mathsf{POR\text{-}LZG}_{\mathsf{egl}}(\mathcal{N})$. If a is a local action, then $\mathsf{Z}'|_{<r} \subseteq \mathsf{Z}|_{<r}$.*

Proof. Let v' be a valuation in Z'. By the pre-property of transitions on zones (Lemma 4.6), we know that there exists a valuation v in Z such that $(q, \mathsf{v}) \xrightarrow{a} \xrightarrow{\delta} (q', \mathsf{v}')$. By commutativity of transitions in local time semantics, we can commute the delay transitions in processes whose $\mathsf{groupid}$ is less than r (denoted by $\delta_{<r}$) to the beginning of the sequence to obtain a sequence of the form

$$(q, \mathsf{v}) \xrightarrow{\delta_{<r}} (q, \mathsf{v}_1) \xrightarrow{a} \xrightarrow{\delta_{\geq r}} (q_1, \mathsf{v}')$$

where $\delta_{\geq r}$ is the local delay in processes with $\mathsf{groupid}$ at least r. Since Z is local-time-closed, we have $\mathsf{v}_1 \in \mathsf{Z}$. We observe that $\mathsf{v}_1|_{<r} = \mathsf{v}'|_{<r}$. As v' was arbitrary, this implies that $\mathsf{Z}'|_{<r} \subseteq \mathsf{Z}|_{<r}$. □

We now introduce the notion of $\mathsf{nextglobal}(q, \mathsf{Z}, r)$ which gives an over-approximation of the set of first global actions on paths from (q, Z, r) in the $\mathsf{POR\text{-}LZG_{egl}}$. Consider a path σ starting from the node (q, Z, r) in the $\mathsf{POR\text{-}LZG_{egl}}$. Recall that only the local actions of processes A_i for $i \geq r$ can be executed before a global action is executed. Therefore, such a global action c must satisfy two conditions:

- there must be an outgoing transition c from q_i, for all $i < r$;
- a transition c must be reachable by a sequence of local actions of $\mathsf{groupid} = i$, for all $i \geq r$.

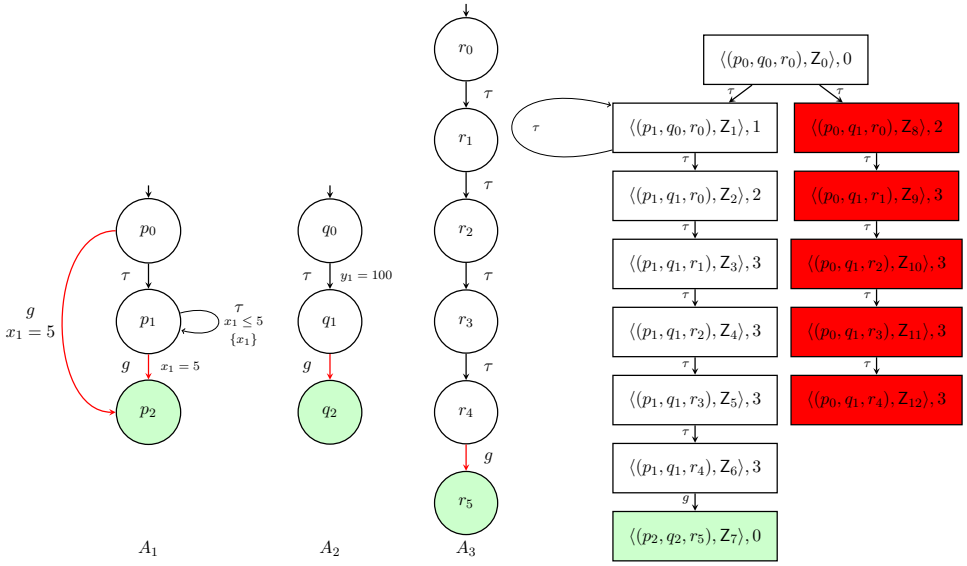
Formally, we can define $\mathsf{nextglobal}(q, \mathsf{Z}, r)$ using the notion of $\mathsf{gl}(i, q, r)$ introduced in Definition 8.12.

Definition 8.13. $\mathsf{nextglobal}(q, \mathsf{Z}, r) = \bigcap_{A_i \in Proc} \mathsf{gl}(i, q, r)$.

If no action in the set $\mathsf{nextglobal}(q, \mathsf{Z}, r)$ is feasible from some node reachable from (q, Z, r), then we can conclude that no outgoing path from (q, Z, r) can be accepting. We prove this in Lemma 8.26. Let $\mathsf{sync}_{<r}$ denote the constraint $t_1 = t_2 = \cdots = t_{r-1}$ that synchronizes the reference clocks of processes 1 to $r-1$. For a global action c and a state q, we let $g^{c,q}_{<r}$ denote the conjunction of the guards of processes $1, \ldots, r-1$ on their transitions on c from q. Recall that we only consider deterministic automata. We let $g^{c,q}_{<r} = false$ if an automaton $i < r$ has no transition on c from q.

Lemma 8.26. *Let $\mathcal{N}$ be an extended global-local system, and let (q, Z, r) be a node of $\mathsf{POR\text{-}LZG_{egl}}(\mathcal{N})$. Let (q, Z, r) be a node of $\mathsf{POR\text{-}LZG_{egl}}(\mathcal{N})$ such that for all actions $c \in \mathsf{nextglobal}(q, \mathsf{Z}, r)$, $\mathsf{Z} \wedge \mathsf{sync}_{<r} \wedge g^{c,q}_{<r} = \emptyset$, Then, a path from (q, Z, r) in $\mathsf{POR\text{-}LZG_{egl}}(\mathcal{N})$ cannot contain a global action.*

Proof. Since $\mathsf{Z} \wedge \mathsf{sync}_{<r} \wedge g^{c,q}_{<r} = \emptyset$ for all actions c in $\mathsf{nextglobal}(q, \mathsf{Z}, r)$, we know that no global action is enabled from (q, Z, r).



(a) Network $\mathcal{N}_3$ (b) $\mathsf{POR\text{-}LZG}_{\mathsf{egl}}(\mathcal{N}_3)$

Figure 8.2: An example that illustrates the effect of Optimization 2 of global-local POR

Let $(q, \mathsf{Z}, r) \xrightarrow{a} (q', \mathsf{Z}', r')$ be a local action in $\mathsf{POR\text{-}LZG}_{\mathsf{egl}}(\mathcal{N})$. By definition of global-local POR, we know that this local action has $\mathsf{groupid}$ at least r. Observe that the components $1, \ldots, r-1$ of q and q' are the same. From Lemma 8.25, we know that $\mathsf{Z}' \wedge \mathsf{sync}_{<r'} \wedge g^{c,q'}_{<r'} = \emptyset$. This implies that no global action is enabled from (q', Z', r').

By repeating this reasoning, we can see that a path from (q, Z, r) cannot contain a global action. □

As a consequence of Lemma 8.26, we get Optimization 2.

Optimization 2. Let $\mathcal{N}$ be an extended global-local system $\mathcal{N}$, and let (q, Z, r) be a node of $\mathsf{POR\text{-}LZG}_{\mathsf{egl}}(\mathcal{N})$ such that for all actions a in $\mathsf{nextglobal}(q, \mathsf{Z}, r)$ we have $\mathsf{Z} \wedge \mathsf{sync}_{<r} \wedge g_a = \emptyset$. Then, by Lemma 8.26, (q, Z, r) need not be stored and its successors need not be explored.

Example for Optimization 2: Consider the network $\mathcal{N}_3$ given in Figure 8.2a. The network consists of three processes A_1, A_2 and A_3. Observe that the accepting state of $\mathcal{N}_3$, (p_2, q_2, r_5), is marked in green. Further, note that the global actions are denoted in red. The POR-zone graph of $\mathcal{N}_3$ is given in Figure 8.2b.

The nodes marked in red will be removed by Optimization 2. In each of these nodes, there is a common global action in the future, but no global action can be enabled from the node or any of its successors. For instance, consider the node $\langle (p_0, q_1, r_0), \mathsf{Z}_8, 2 \rangle$ of $\mathsf{POR\text{-}LZG}_{\mathsf{egl}}(\mathcal{N}_3)$. The global action

g is an outgoing action from the state p_0 in A_1 and the state q_1 in A_2, and is reachable by local actions from r_0 in A_3. So, this node will not be eliminated by Optimization 1. However, observe that the set $\mathsf{Z}_8 \wedge (t_1 = t_2) \wedge (x_1 = 5)$ is empty since Z_8 implies $(\widetilde{x}_1 = \widetilde{y}_1) \wedge (t_2 - \widetilde{y}_1 \geq 100)$ (recall that $x_1 = 5$ is translated as $t_1 - \widetilde{x}_1 = 5$ in the offset settings). Therefore, the action g is never enabled from this node, even if we follow the sequence of local actions in A_3. In a similar way, all the nodes highlighted in $\mathsf{POR\text{-}LZG}_{\mathsf{egl}}(\mathcal{N}_3)$ as shown in Figure 8.2b will not be explored due to Optimization 2.

We now present a revised version of the successor computation procedure, $\mathsf{Next}^*_{\mathsf{gl}}$ (Algorithm 10) with the two optimizations incorporated into it. Optimization 1 is reflected in line 8, and Optimization 2 is given by line 9 of the algorithm.

Algorithm 10 $\mathsf{Next}^*_{\mathsf{gl}}(q, \mathsf{Z}, r)$

Input: A node (q, Z, r) of the POR-zone graph of a network of timed automata $\mathcal{N}$
Output: A set of successors of (q, Z, r).

```
1:  Succ := ∅
2:  for every transition (q, Z) -a-> (q', Z') do
3:      if a is global then
4:          Add (q', Z', 0) to Succ
5:      else
6:          r' = groupid(a)
7:          if (r' ≥ r) ∧
8:             (∀i, j gl(i, q', r') ∩ gl(j, q', r') ≠ ∅) ∧
9:             not (∀_{c∈nextglobal(q',Z',r')}. Z' ∧ sync_{<r'} ∧ g_c = ∅) then
10:
11:             Add (q', Z', r') to Succ
12: return Succ
```

Client-server POR

In this section, we discuss an optimization to the $\mathsf{Next}_{\mathsf{ecs}}$ procedure that exploit properties of extended client-server systems. Recall that in an extended client-server system, each client process needs to execute a communication action with the server just before reaching an accepting state. As a consequence, if we can deduce that on all paths from a given node of $\mathsf{POR\text{-}LZG}_{\mathsf{ecs}}(\mathcal{N})$ there cannot be a communication action, then we can avoid further exploration from this node.

Let the network of timed automata $\mathcal{N} = \langle S, C_1, C_2, \cdots C_k \rangle$ be a client-server system. Let q be a state. By $\mathsf{comm}(q, 0)$ we denote the set of communication actions of the server that label outgoing transitions from q.

We also introduce a more complicated notion $\mathsf{comm}^*(q, r)$, for rank $r > 0$. It gives an over-estimation of the set of communication actions that can be executed by processes of groupid equal to r. Note that from a node of rank r, either all the actions are explored (if $r = 0$) or only actions of processes whose groupid is r are explored till the next communication with a server (if $r \neq 0$).

Definition 8.14. Let q be a state of $\mathcal{N}$ and let $r > 0$ be a rank. A communication action c belongs to $\mathsf{comm}(q, 0)$ if there is an outgoing server transition labeled by c from q. A communication action c belongs to $\mathsf{comm}^*(q, r)$ if there is a sequence of client actions with groupid equal to r, leading to a state q' with an outgoing transition labeled c.

Lemma 8.27. *Let $\mathcal{N}$ be an extended client-server system, and let (q, Z, r) be a node of $\mathsf{POR\text{-}LZG}_{\mathsf{ecs}}(\mathcal{N})$ with $r > 0$. If $\mathsf{comm}^*(q, r) \cap \mathsf{comm}(q, 0) = \emptyset$, then there can be no communication action on a path from (q, Z, r) in $\mathsf{POR\text{-}LZG}_{\mathsf{ecs}}(\mathcal{N})$.*

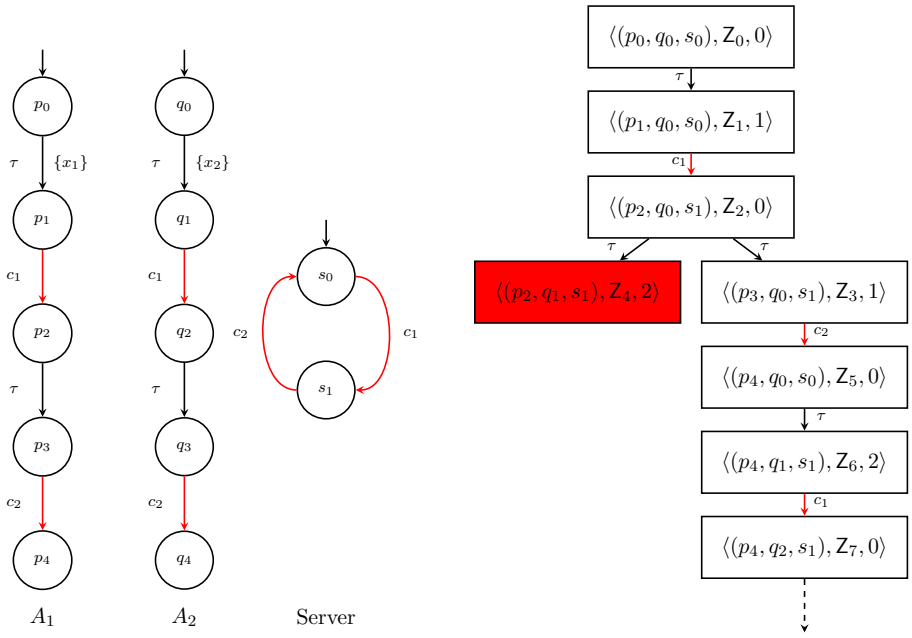
Proof. From the node (q, Z, r), only actions of processes with groupid equal to r are explored. Recall that $\mathsf{comm}(q, r)$ denotes the set of all communication actions that are reachable by a sequence of local actions of processes with groupid equal to r. Since $\mathsf{comm}^*(q, r) \cap \mathsf{comm}(q, 0) = \emptyset$, we know that no such action is enabled from the current location of the server. This means that no communication action is enabled from (q, Z, r) nor can it become enabled by only executing client actions with groupid equal to r in $\mathsf{POR\text{-}LZG}_{\mathsf{ecs}}(\mathcal{N})$. □

As a corollary of Lemma 8.27, we immediately get Optimization 3.

Optimization 3. Let $\mathcal{N}$ be an extended client-server system. Let (q, Z, r) be a node of $\mathsf{POR\text{-}LZG}_{\mathsf{ecs}}(\mathcal{N})$ such that $\mathsf{comm}^*(q, r) \cap \mathsf{comm}(q, 0) = \emptyset$. Then, by Lemma 8.27, (q, Z, r) need not to be stored, and its successors need not be explored.

Example for Optimization 3: Consider the network $\mathcal{A}$ given in Figure 8.3a.

The network consists of two client processes A_1 and A_2, and a server process S. Note that the communication actions of the network are denoted in red. A part of the POR-zone graph of $\mathcal{A}$ is given in Figure 8.3b. Consider the node $\langle (p_2, q_1, s_1), Z_4, 2\rangle$ of $\mathsf{POR\text{-}LZG}_{\mathsf{ecs}}(\mathcal{A})$ that is marked in red. Observe that c_2 is the only communication action that is reachable from q_1 via (exclusively) local actions. This is expressed as $\mathsf{comm}^*((p_2, q_1, s_1), 2) = \{c_1\}$. However, the set of outgoing actions from the current state s_1 of server is $\{c_2\}$, i.e., $\mathsf{comm}((p_2, q_1, s_1), 0) = \{c_2\}$. Since $\mathsf{comm}^*((p_2, q_1, s_1), 2) \cap \mathsf{comm}((p_2, q_1, s_1), 0) = \emptyset$, we know that there is no common communication action in the future from this node or any of its successors. Therefore, the node (p_2, q_1, s_1, Z_4) will be removed by Optimization 3.



(a) Network $\mathcal{A}$ (b) POR-zone graph of $\mathcal{A}$

Figure 8.3: An example that illustrates the effect of Optimization 3 of Client-server POR

We now present the revised Next procedure for extended client-server systems with Optimization 3 as reflected in line 4 of the algorithm.

Algorithm 11 $\mathsf{Next}^*_{\mathsf{cs}}(q, \mathsf{Z}, r)$

Input: A node (q, Z, r) of $\mathsf{POR\text{-}LZG}_{\mathsf{ecs}}(\mathcal{N})$ of a client-server network $\mathcal{N}$
Output: A set of successors of $9q, \mathsf{Z}, r)$

```
1:  Succ := ∅
2:  for every transition (q, Z) -a-> (q', Z') do
3:      if r = 0 or
4:       (there is C_i ∈ dom(a) with
5:       groupid(C_i) = r, and comm*(q, r) ∩ comm(q, 0) ≠ ∅) then
6:          if S ∈ dom(a) then
7:              r' = 0
8:          else
9:              r' = i for some C_i ∈ dom(a)
10:         Add (q', Z', r') to Succ
11: return Succ
```

Chapter 9

Experiments

In this thesis, we have proposed two different solutions to address the challenge of state-space explosion for the reachability problem for networks of timed automata. The first solution, based on the computation of the local sync graph (Definition 5.6) of the network, while the second solution involves applying partial order reduction to a finite truncation of the local zone graph of the network. A prototype of both of these algorithms has been implemented in the tool TChecker [HP19]. In this chapter, we discuss the experiments with these implementations on standard benchmarks for timed automata. We compare the performance of these implementations with the standard UPPAAL reachability algorithm, which is the state-of-the-art solution to solve the reachability problem for timed automata.

In the first part of this chapter, we focus on the implementation of Algorithm 3 discussed in Chapter 5. The second part of this chapter is concerned with the implementation of the POR-reachability procedure (Algorithm 5) that was proposed in Chapter 8. We compare the results of running the implementations of these procedures alongside the results for the classical UPPAAL reachability algorithm. All the three algorithms solve the reachability problem for a network of timed automata by computing a transition system and exploring it to search for an accepting state. The classical reachability algorithm computes a transition system called the global zone graph of the network, while Algorithm 3 computes the local sync graph of the network (see Definition 5.6) and Algorithm 5 computes the POR-zone graph (see Section 8.2) of the network.

For both our implementations, we first introduce some toy models (that are essentially simple networks that are easy to understand), and illustrate the working of our implementations on these simple networks. Equipped with this understanding, we discuss the results of running these implementations on bigger models that constitute the standard benchmarks for timed automata. We also give a comparison of their performance to that of the classical UPPAAL reachability algorithm provided in TChecker and provide tables

that display the exhibit the results.

As a metric of the performance of the algorithms, for each network of timed automata that we run experiments on, we consider three parameters for comparison: the number of visited nodes and stored nodes in the transition system computed by the algorithm, and the runtime of the algorithm. We analyse the performance of our implementations and wherever applicable, we articulate the reasons for the gains, or lack thereof, of our implementation with respect to the standard procedure.

Remark (Runtime of the algorithm)**.** *TChecker runs 2 threads for every execution: a computation thread and a garbage collection thread. The user time sums the running time of both threads, whereas the total time is the maximum of the two times. For the purposes of experiments in this chapter, we have taken the total time as an estimate of the running time.*

9.1. Efficient reachability testing using local sync graphs

In this section, we discuss the implementation of Algorithm 3 that solves the reachability problem for a network of timed automata by the construction of its local sync graph. We will refer to this implementation as the local sync graph implementation (LSG-implementation). We discuss the performance of the LSG-implementation, and compare its performance with the implementations of the classical UPPAAL reachability algorithm provided in TChecker.

We first introduce a toy-model for which the LSG-implementation computes local sync graphs that are an order of magnitude smaller than the global zone graphs computed by the standard reachability procedure. Using this toy-model, we will analyse the working of LSG-implementation by studying the local sync graph of the model that it generates. We compare and contrast the global zone graph and the local sync graph of the toy model and explain the source of gains for the model.

9.1.1 Toy model for LSG implementation

Our toy model is given by a network of timed automata $\mathcal{N}$, with n identical processes, as depicted in Figure 9.1. Note that the actions b and c are global actions, while all other actions are local actions. Each automaton A_i executes local action a_i, after which one of the global actions, b or c is executed.

Table 9.1 presents results of our experiments on instances of the toy model with different number of processes. From Table 9.1, we can observe an order of magnitude gains for our toy models. We now clarify the source of these gains by explaining the working of the LSG implementation on this

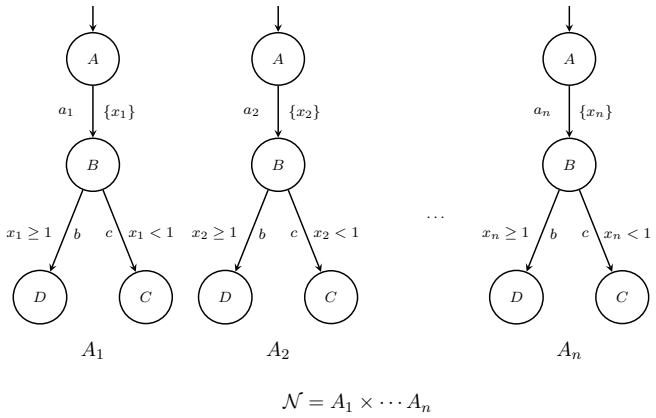
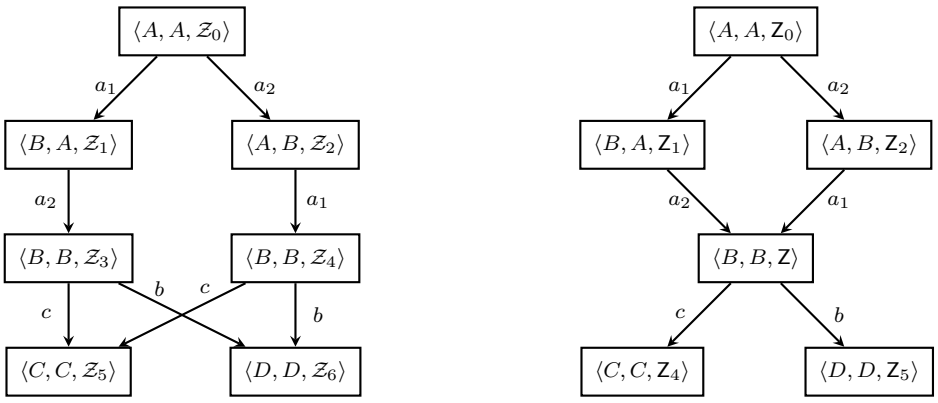


Figure 9.1: A toy model to explain the working of the LSG implementation

model. We do this by considering the instance of $\mathcal{N}$ with 2 processes, A_1 and A_2. The global zone graph and the local sync graph for this network are given in Figures 9.2a and 9.2b, respectively.



(a) Global zone graph of $\mathcal{N}$ (b) Local sync graph of $\mathcal{N}$

Figure 9.2: Zone graphs of network $\mathcal{N}$ given in Figure 9.1

In both A_1 and A_2, there is a local action a_i followed by a global action b or c. Observe that when A_1 and A_2 perform local actions a_1 and a_2, doing a_1a_2 or a_2a_1 leads to the same control-state of the automaton. From Figure 9.2a, we can see that in the global zone graph of $\mathcal{N}$, the paths a_1a_2 and a_2a_1 lead to different zones $\mathcal{Z}_1$ and $\mathcal{Z}_2$, in the global zone graph. As a result, we have two copies of the node with the control state (B, B), one per each interleaving of the local actions. On the other hand, the two paths result in the same zone Z (containing both $\mathcal{Z}_1$ and $\mathcal{Z}_2$) in the local sync graph (see Figure 9.2b). In this way, our new reachability procedure is able to construct a local zone graph that is an order of magnitude smaller than the global zone graph of the network, as can be observed from Table 9.1.

Models	Global ZG			Local SG		
(# processes)	visited	stored	time	visited	stored	time
2	7	7	0.033	6	6	0.030
3	18	18	0.078s	10	10	0.049s
4	67	67	0.283s	18	18	0.322s
5	328	328	1.360s	34	34	0.203s
8	109,603	109,603	5m19s	258	258	2.267s

Table 9.1: Experimental results for the toy model given in Figure 9.1. The table gives the statistics of the global zone graph and the local sync graph with different number of processes.

Remark. *Observe that we have two global actions from the state B of each component, namely b with a lower bound guard, and c with an upper bound guard. The purpose of these two actions is to force the $\mathfrak{a}_{\preccurlyeq_{LU}}$ subsumption to differentiate between the zone reached by the different interleavings of local actions a_1 and a_2. Had there been only one global action, when using the $\mathfrak{a}_{\preccurlyeq_{LU}}$ subsumption, one zone gets subsumed by the other zone since there is no active lower bound (or upper bound) from the resultant state. We remark that if we use $\mathfrak{a}_M$-subsumption while computing the global zone graph, we do not need the two global actions b and c in the model to differentiate the zones $\mathcal{Z}_1$ and $\mathcal{Z}_2$.*

9.1.2 Benchmark models

In this section, we discuss the results of the experiments done using the LSG-implementation for some of the classical benchmarks for timed automata. We compare our implementation with two implementations of the standard reachability procedure based on the exploration of the global zone graph of the network: TChecker and UPPAAL [LPY97, BDL+06], the state-of-the-art verification tool for timed automata. All the three implementations use a breadth-first search with subsumption. Note that in the case of local sync graph, the subsumption used is the $\sqsubseteq^{\mathfrak{a}LU}_{\mathsf{sync}}$ subsumption (see Definition 5.5). Table 9.2 presents results of our experiments with standard models from the literature (except *Parallel* that is a model we have introduced). The model *Parallel* as presented in Figure 9.3, is a network consisting of n identical processes $A_1, \cdots, A_n$, that do not communicate with one another, and a centralized process *Lock* that communicates with each process A_i. (One can see that *Parallel* is a client-server system, as defined in Section 7.3.) The actions *acquire* and *release* are communication actions, while all other actions are local actions.

Local sync graphs yield no gain on 3 standard examples (which are not given in Table 9.2): *CSMA/CD* [TY01], *FDDI* [DOTY95] and *Fischer* [TY01]. In these models, the three algorithms visit and store the same

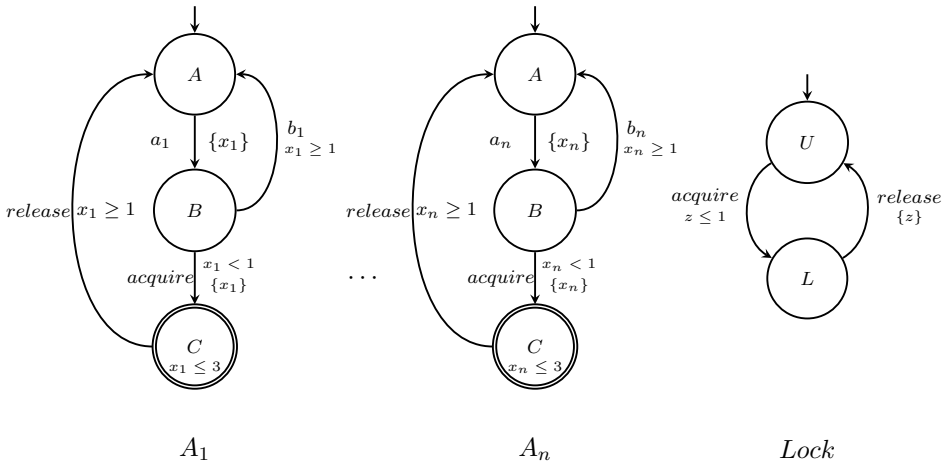


Figure 9.3: Parallel model

number of nodes. The reason is that for *CSMA/CD* and *FDDI* replacing each local zone Z in the local zone graph by its set of synchronized valuations $sync(\mathsf{Z})$ yields exactly the global zone graph. In the third model, *Fischer*, each control state appears at most once in the global zone graph. So there is no hope to achieve any gain with our technique. This is due to the fact that doing *ab* or *ba* results in two different control states in the automaton. As a consequence, *Fischer* is out of the scope of our technique.

In contrast, we observe significant improvements on other standard models (Table 9.2). Observe that due to subsumption, the order in which nodes are visited impacts the total number of visited nodes. UPPAAL and our prototype TChecker (Global ZG column) may not visit the same number of nodes despite the fact that they implement the same algorithm. In our prototype we use the same order of exploration for Global ZG and Local ZG. *CorSSO* [YW06] and *Critical region* [MPS11] are standard examples from the literature. *Dining Philosophers* [LNZ05] is a modification of the classical problem where a philosopher releases her left fork if she cannot take her right fork within a fixed amount of time. We observe an order of magnitude gains for most of these four models. The reason is that, as we illustrated using our toy model in Section 9.1.1, in most states, when two processes can perform actions a and b, doing ab or ba leads to the same control-state of the automaton. Hence, a difference between ab and ba (if any) is encoded in distinct zones $\mathcal{Z}_{ab}$ and $\mathcal{Z}_{ba}$ obtained along these two paths in the global zone graph. In contrast, the two paths result in the same zone Z (containing both $\mathcal{Z}_{ab}$ and $\mathcal{Z}_{ba}$) in the local sync graph. In consequence, our approach that combines the local zone graph and abstraction using synchronized zones is very efficient in this situation.

Models	UPPAAL			Global ZG			Local ZG		
(# processes)	visited	stored	sec.	visited	stored	sec.	visited	stored	sec.
CorSSO 3	64378	61948	1.48	64378	61948	1.41	1962	1962	0.05
CorSSO 4	timeout			timeout			23784	23784	0.69
CorSSO 5	timeout			timeout			281982	281982	16.71
Critical reg. 4	78049	53697	1.45	75804	53697	2.27	44490	28400	2.40
Critical reg. 5	timeout			timeout			709908	389614	75.55
Dining Phi. 7	38179	38179	34.61	38179	38179	7.28	2627	2627	0.32
Dining Phi. 8	timeout			timeout			8090	8090	1.65
Dining Phi. 9	timeout			timeout			24914	24914	7.10
Dining Phi. 10	timeout			timeout			76725	76725	30.20
Parallel 6	11743	11743	4.82	11743	11743	1.09	256	256	0.02
Parallel 7	timeout			timeout			576	576	0.04
Parallel 8	timeout			timeout			1280	1280	0.11

Table 9.2: Experimental results obtained by running UPPAAL and our prototype TChecker (Global ZG and Local ZG) on a MacBook Pro 2013 with 4 2.4GHz Intel Core i5 and 16 GB of memory. The timeout is 90 seconds. For each model we report the number of concurrent processes.

9.2. POR-implementation in TChecker

In this section, we discuss the experiments performed with the implementation of the partial order reduction procedure implemented in TChecker [HP19]. We discuss the performance of the POR-reachability algorithm as implemented in TChecker, and compare its performance with two other implementations: the classical UPPAAL reachability algorithm and the LSG-implementation discussed in Section 9.1. We recall that our POR-reachability algorithm computes a transition system called the POR-zone graph (denoted as $\mathsf{POR\text{-}LZG}_r$), discussed in detail in Section 8.2.

For a comparison of the three procedures, for each model, we compile the results of experiments in a table. For each procedure, the table gives the number of visited nodes and stored nodes in the transition system computed by the procedure, and the runtime of the procedure, for the model with different number of processes.

Remark. *We run the algorithm with the default values of parameters for smaller models, which produce zone graphs of order 1000 nodes. For larger zone graphs, we increase the values of these parameters to block-size* $= 100,000,000$ *and table-size* $= 65,536,000$*, as the size of the models increase.*

Recall that we had proposed two variants of the POR implementation - namely, global-local POR (Section 8.3) and client-server POR (Section 8.4), targeted at two classes of networks of timed automata, global-local systems and client-server systems, respectively. Thus, each model can be classified into one of the following categories, each of which represents a situation that can arise when applying our POR technique:

Type 1. Spread bounded global-local model where POR works

Type 2. Spread bounded global-local model where POR does not work

Type 3. Spread bounded client-server model where POR works

Type 4. Spread bounded client-server model where POR does not work

Type 5. Models with unbounded spread - so, our methods do not apply

We will first present some toy models, one from each of the categories. We discuss the performance of our POR-implementation on these toy models, and explain the reasons for the gains or lack of gains observed while running the implementation on the model. By doing this, we hope to present and distinguish the situations, favourable and unfavourable for our methods. We then discuss the working of our POR-implementation on some of the classical benchmarks for timed automata, and explain the gains (or lack of gains) for our algorithm on these models.

9.3. Toy models

In this section, we will introduce some toy models and discuss the results of applying the POR-implementation on these networks. We describe, in some detail, how our procedure works on these toy models. Through these examples, we hope to convey to the reader, an intuition of the working of our implementation and the source of gains or lack thereof, for these models.

We will consider five toy models, one from each of the categories given in the classification given above.

9.3.1 Type 1: Global-local toy model where POR works

Here, we present a toy global-local model for which our global-local POR method is much more efficient than the standard reachability procedure. The model is given by a network of timed automata $\mathcal{N}$, with n identical processes, as depicted in Figure 9.4. Note that the action g is a global action, while all other actions are local actions. Each automaton in this network is of the form of a chain of local actions with a global action at the end. Thus, each process is free to execute its sequence of local actions independently, and in the end jointly execute a global action.

Table 9.3 presents the result of experiments on instances of the model in Figure 9.4 with different number of processes.

From the definition of tame model (see Definition 7.3) , we can see that the model in Figure 9.4 is a tame model. As a consequence, from Lemma 7.1, it immediately follows that it is 0-spread. We run the global-local POR-reachability algorithm with the parameter spread set to 0, for this model.

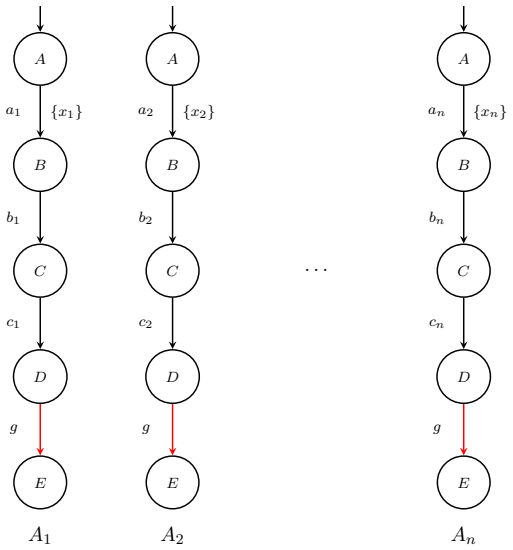


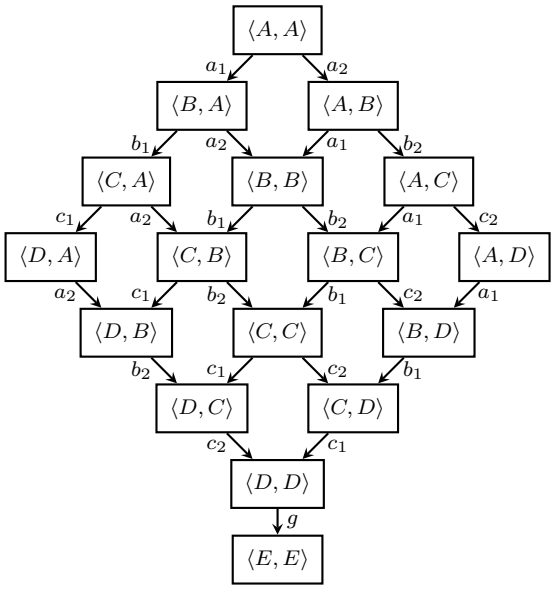
Figure 9.4: A toy global-local model for which our global-local POR obtains good gains

Models	Global ZG			Local sync graph			POR-zone graph		
(# processes)	visited	stored	time	visited	stored	time	visited	stored	time
2	17	17	0.078	17	17	0.077	8	8	0.040
3	65	65	0.319	65	65	0.349	11	11	0.044s
4	257	257	1.745s	257	257	2.295s	14	14	0.062s
5	1,025	1,025	10.264s	1,025	1,025	10.159s	17	17	0.091s
8	65,537	65,537	14m51s	65,537	65,537	14m52s	26	26	0.113s

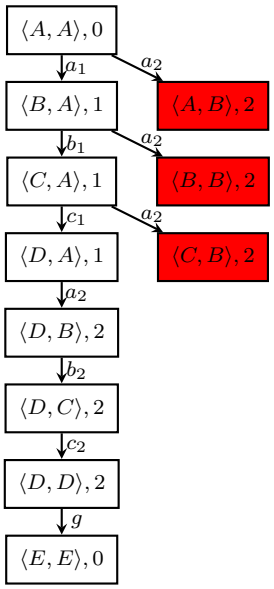
Table 9.3: Experimental results for the toy global-local model with different number of processes. Experiments for models with up to 4 processes were run with default values of parameters. For the model more than 4 processes, the computation of global zone graphs and local sync graphs were done with values of parameters set to block-size $= 100,000,000$ and table-size $= 65,536,000$.

We now explain the working of the global-local POR-reachability algorithm on this model. We do this by considering the instance of $\mathcal{N}$ with 2 processes, A_1 and A_2. The global zone graph and the POR-zone graph for this network are given in Figures 9.5a and 9.5b, respectively. Observe that A_1 has a sequence of local actions $a_1b_1c_1$ followed by a global action g. Similarly, A_2 has a sequence of local actions $a_2b_2c_2$ followed by the global action g. From Figure 9.5a, we can see that the global zone graph of $\mathcal{N}$ explores all the possible interleavings of these sequences of local actions. As a result, we have a diamond composed of these local actions in the global zone graph.

On the other hand, in $\mathsf{POR\text{-}LZG}_r(\mathcal{N})$, we explore only one interleaving of



(a) Global zone graph



(b) POR-zone graph

Figure 9.5: Global zone graph and POR-zone graph (without the zone information) of the instance of model in Figure 9.4 with 2 processes

this sequence of local actions, as can be seen in Figure 9.5b. In $\mathsf{POR\text{-}LZG}_r(\mathcal{N})$, all the local actions of A_1 are explored before exploring the local actions of A_2. We will now see how this is accomplished. The initial node of $\mathsf{POR\text{-}LZG}_r(\mathcal{N})$ has rank 0, which means that local actions of all the processes are explored from this node. Consider the node (A, B) reached by exploring the local action a_2 in A_2. Recall that this node has rank 2. Using Optimization 1 for global-local POR, we are able to conclude that no global action is possible from this node or any of its successors in $\mathsf{POR\text{-}LZG}_r(\mathcal{N})$. Thus, the exploration of this node is not useful, as far as reachability is concerned. Therefore, we avoid the exploration of this node and its successors. In a similar manner, until we reach the local state (D, A) in $\mathsf{POR\text{-}LZG}_r(\mathcal{N})$, we are able to infer that exploration of local actions in A_2 are useless w.r.t. reachability. So, a local action of process A_2 is explored only when there are no more local actions left to be explored in A_1.

In this way, our global-local POR method is able to construct a POR-zone graph that is linear in the size of the network, while the global zone graph is exponential in the size of the network.

9.3.2 Type 2: Global-local toy model where POR does not work

In this section, we present a toy model that illustrates a typical situation where our global-local POR technique does not fare well. The toy model is given by the network of automata given by the network $\mathcal{N}$, as given in Figure 9.6. As was the case for the toy model in Section 9.3.1, we can see that the model in Figure 9.6 is a tame model and it immediately follows that it is 0-spread.

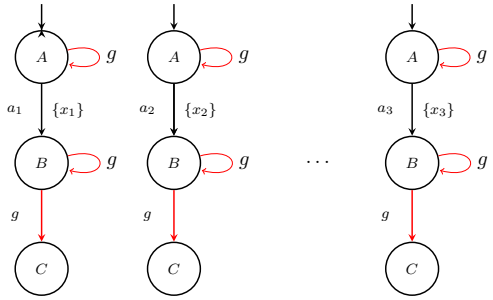


Figure 9.6: A toy global-local model for which global-local POR does not give gains.

Models	Global ZG			Local SG			POR-zone graph		
(# processes)	visited	stored	time	visited	stored	time	visited	stored	time
2	9	9	0.435s	9	9	0.989s	12	9	0.058s
3	27	27	0.141s	27	27	0.148s	34	27	0.119s
4	81	81	0.427s	81	81	0.448s	96	81	0.320s
5	243	243	1.566s	243	243	1.611s	274	243	0.896s
8	6561	6561	56.498s	6561	6561	56.121s	6816	6561	28.731s

Table 9.4: Experimental results obtained by running prototype TChecker on the toy global-local model given in Figure 9.6. Experiments for models of up to 4 processes were run with default values of parameters. For the model with 8 processes, experiments were done with values of parameters set to block-size $= 100,000,000$ and table-size $= 65,536,000$.

Table 9.4 presents the results of experiments on instances of the model in Figure 9.6 with different number of processes. The POR-implementation was run with spread set to 0.

To understand the lack of gains for this model, we first consider a simpler network $\mathcal{A}$ given in Figure 9.7. Consider the POR-zone graph of $\mathcal{A}$ produced by our global-local POR-algorithm without the optimizations, given in Figure 9.8b. Comparing the global zone graph of $\mathcal{A}$, $\mathsf{ZG}(\mathcal{A})$ (Figure 9.8a), and $\mathsf{POR\text{-}LZG}'_r(\mathcal{A})$, we can see that even though we have far fewer interleavings in $\mathsf{POR\text{-}LZG}'_r(\mathcal{A})$ as compared to the global zone graph, the number of states is the same in both the transition systems. Thus, if we consider the number

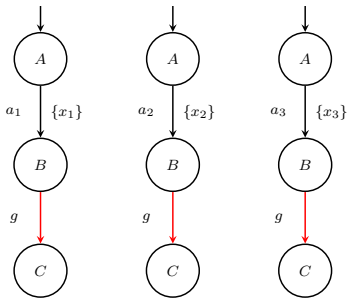
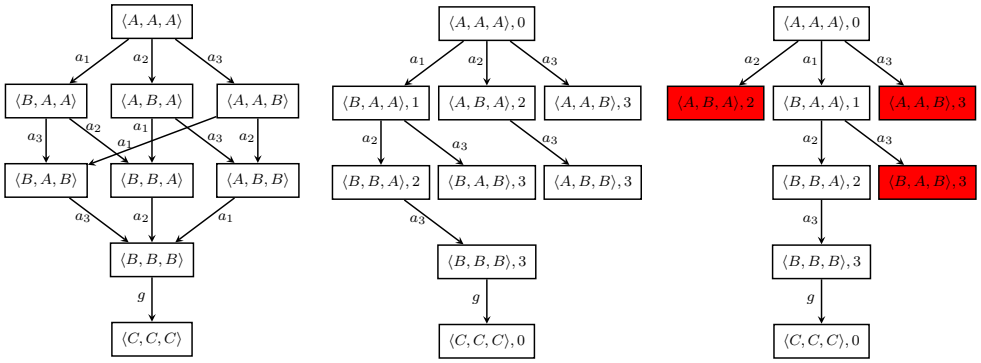


Figure 9.7: Network $\mathcal{A}$

of states as the parameter of comparison, our POR-zone graph is no better than the global zone graph.

Next, we consider the POR-zone graph of $\mathcal{A}$, $\mathsf{POR\text{-}LZG}_r(\mathcal{A})$, computed by the optimized version of our POR-procedure, given in Figure 9.8c. (Note that the red nodes are not part of $\mathsf{POR\text{-}LZG}_r(\mathcal{A})$.) Recall that Optimization 1 was used to eliminate nodes of the POR-zone graph from which no global action is possible in the future. Observe that Optimization 1 helps in eliminating a lot of nodes from $\mathsf{POR\text{-}LZG}_r(\mathcal{A})$ that are irrelevant to reachability.



(a) $\mathsf{ZG}(A)$ (b) Unoptimized $\mathsf{POR\text{-}LZG}_r(\mathcal{A})$ (c) Optimized $\mathsf{POR\text{-}LZG}_r(A)$

Figure 9.8: Global zone graph and POR-zone graph generated by the POR-implementation with and without Optimization 1, of network $\mathcal{A}$ as given in Figure 9.7. We have omitted the zone information for all the zone graphs.

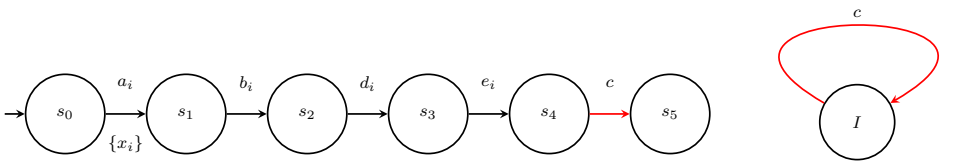
A crucial reason why Optimization 1 worked was that the initial state did not have any global outgoing actions. Had there been outgoing global action g on all the states, Optimization 1 would not have removed any node of the POR-zone graph. This is, in fact, the situation in the model in Figure 9.6 - there is a self-loop on the global action g on the states A and B. As a result, in this network, the global action g is enabled from all configurations of states. Thus, in this situation, Optimization 1 is no longer useful. As a consequence, we would not get any gains due to our global-local POR

implementation.

Remark. *Despite the lack of gains with our global-local POR-method, observe that in Table 9.4, the runtime of the POR-implementation is lesser than that of the standard algorithm and the LSG implementation. This is because the number of interleavings in the POR-zone graph is much lesser than the number of interleavings in the global zone graph and the local sync graph of the model.*

9.3.3 Type 3: Client-server toy model where POR works

In this section, we present a toy client-server model for which our client-server POR method is more efficient than the standard reachability procedure. We discuss the working of client-server POR method on this model and explain the source of gains. The model, as depicted in Figure 9.9, is a network of timed automata $\mathcal{N}$ containing a server process and n (identical) client processes. Note that in the process Client$_i$, the action c (highlighted in red) is a communication with the server, while all other actions are local actions.



(a) Client C_i (b) Server process

Figure 9.9: A toy client-server model for which our client-server POR gives good gains

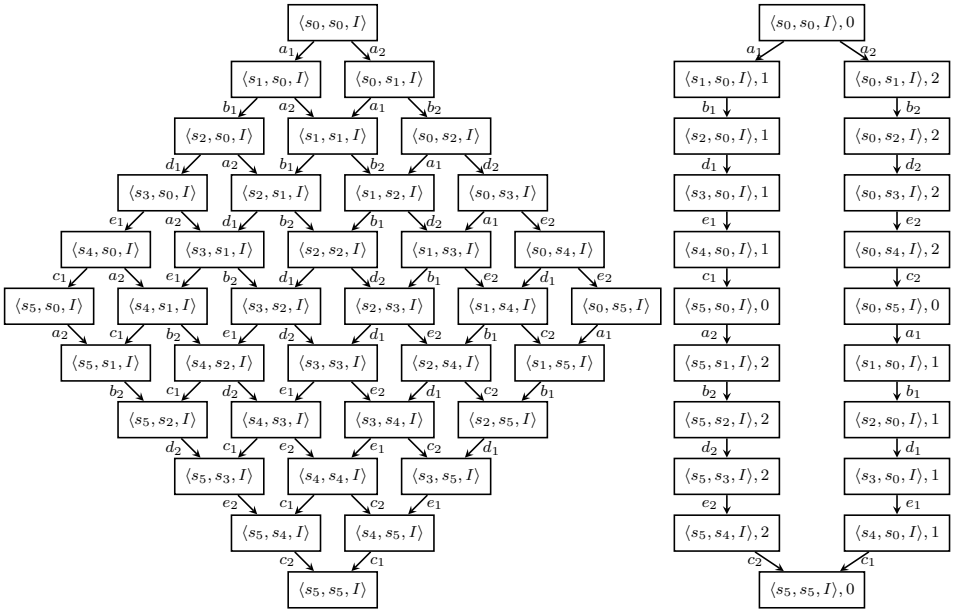
Since the network has no timing constraints, we can convert any run to a run in which each action is executed in 0 time. Therefore, we can conclude that the model has spread 0. We consider the global zone graph and POR-zone graph of spread 0, of the toy client-server model. The client-server POR-reachability algorithm is run with the parameter spread set to 0. Table 9.5 presents the results of experiments on instances of the model in Figure 9.9 with different number of clients.

We now explain the working of the client-server POR-reachability algorithm by illustrating how our procedure works on an instance of this toy model. For simplicity, we consider the instance of the model with just 2 clients. Figure 9.10a gives the global zone graph and Figure 9.10b gives the POR-zone graph of this network.

Observe that there is a sequence of actions $a_1b_1d_1e_1c$ in A_1 and $a_2b_2d_2e_2c$ in A_2. We can see, from Figure 9.10a, that all the interleavings of the sequences $a_1b_1d_1e_1c$ and $a_2b_2d_2e_2c$ are explored in the global zone graph. As a consequence, we are forced to explore a lot of intermediate states in

Models	Global ZG			Local SG			POR-zone graph		
(# clients)	visited	stored	time	visited	stored	time	visited	stored	time
2	36	36	0.590	36	36	0.152	20	20	0.078
3	216	216	1.286	216	216	1.271	56	56	0.182
4	1296	1296	9.804s	1296	1296	9.606s	144	144	0.444s
5	7776	7776	1m27s	7776	7776	1m15s	352	352	4.159s
8	1,679,616	1,679,616	98m18s	1,679,616	1,679,616	95m10s	4352	4352	25.206s

Table 9.5: Experimental results for the toy client-server model given in Figure 9.9. Experiments for models of up to 4 clients were run with default values of parameters. For the model with 5 and 8 clients respectively, experiments were done with values of parameters set to block-size $= 100,000,000$ and table-size $= 65,536,000$.



(a) Global zone graph (b) POR-zone graph

Figure 9.10: Global zone graph and POR-zone graph (without the zone information) of the instance of model in Figure 9.9 with 2 clients.

the global zone graph, which do not lead to any new accepting states, and therefore do not provide any additional information w.r.t. reachability.

Now, consider the POR-zone graph of this network. Here, we start with the initial node (s_0, s_0, I) whose rank is 0. From this node, we explore all enabled actions. Next, consider a successor of this node in the POR-zone graph - the node (s_1, s_0, I). This node has rank 1. Recall that from a node of rank r, if $r \neq 0$, only actions of process A_r are explored till the next communication with a server. Thus, even though the action a_2 is enabled from the node (s_1, s_0, I), the client-server POR procedure postpones the exploration of a_2 to a later point. Note that from a lot of states (as

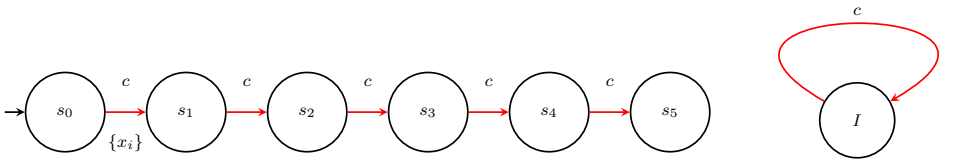
demonstrated above for (s_1, s_0, I)) in the POR-zone graph, the exploration of many actions is postponed, thereby avoiding the exploration of nodes which do not contribute any new information with respect to reachability.

More concretely, thanks to our client-server POR algorithm, once we start exploring the actions of a client, we continue to explore only local actions of that client until we see a communication action. Thus, the client-server POR method ensures that we do not explore all execution orders in $\mathsf{POR\text{-}LZG}_r(\mathcal{N})$. Rather, we explore only two symmetric orders - the sequence of actions of A_1 followed by the sequence of actions in A_2, and the sequence of actions of A_2 followed by the sequence of actions in A_1. In this way, we obtain a POR-zone graph that is much smaller (w.r.t. the number of nodes) than the global zone graph.

9.3.4 Type 4: Client-server toy model where POR does not work

In this section, we present a typical situation where our client-server POR technique does not give us any gains.

Consider a client-server system $\mathcal{A}$, in which every action of each process is a communication action (see Figure 9.11). This means that each node of the POR-zone graph has rank 0. As a consequence, from every node of the POR-zone graph, all the actions will be explored. Thus, there is no reduction in the number of nodes when we compare the POR-zone graph of $\mathcal{A}$, when compared to the global zone graph of $\mathcal{A}$.



(a) Client C_i (b) Server process

Figure 9.11: A toy client-server model $\mathcal{A}$ for which our client-server POR does not give any gains

Models (# clients)	Global ZG			Local SG			POR-zone graph		
	visited	stored	time	visited	stored	time	visited	stored	time
CS-toy-bad-2	36	36	0.024s	36	36	0.021s	36	36	0.023s
CS-toy-bad-3	216	216	0.066s	216	216	0.061s	216	216	0.065s
CS-toy-bad-4	1,296	1,296	0.465s	1,296	1,296	0.409s	1,296	1,296	0.429s
CS-toy-bad-5	7,776	7,776	3.5s	7,776	7,776	3.9s	7,776	7,776	3.9s

Table 9.6: Experimental results for the toy client-server model given in Figure 9.11.

9.3.5 Type 5: Toy model with unbounded spread

We have already presented an example of a network of timed automata that has unbounded spread in Section 7.1. We repeat this example here, so as to illustrate a typical situation that prevents a system from having a bounded spread.

Consider a network $\mathcal{A}$ of two processes, A_1 and A_2, as shown in Figure 9.12.

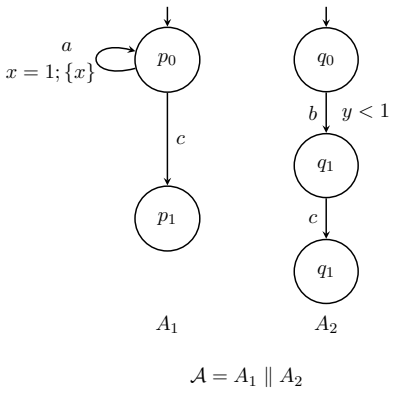


Figure 9.12: Example of a network with unbounded spread

We can show that the run $a^m bc$ of $\mathcal{A}$ is possible in local time semantics and has spread at least $m - 1$, which implies that the spread of $\mathcal{A}$ is not bounded. For a proof, see Lemma 7.1.

9.4. Bigger benchmarks

In this section, we discuss the results of the experiments with our POR-implementation on some of the classical benchmarks for timed automata. We will consider some networks of timed automata inspired from classical benchmarks and classify them into one of the categories of the classification given in page 193. Using the intuition obtained from the working of the POR-algorithm on the toy models, we explain the gains (or lack thereof) for our POR-implementation on these models.

9.4.1 Type 1: Global-local model where POR works

Here, we consider a global-local system for which our global-local POR method turns out to be more efficient than the standard reachability procedure. We consider the global-local variant of the *Fire-alarm* model, a model from the standard benchmarks for timed automata.

Fire-alarm model

Standard model. The standard Fire-alarm model [MWP12] is a network consisting of n processes, referred to as Sensor processes, and a server process. Each process in the model is modelled using a timed automaton as depicted in Figure 9.13.

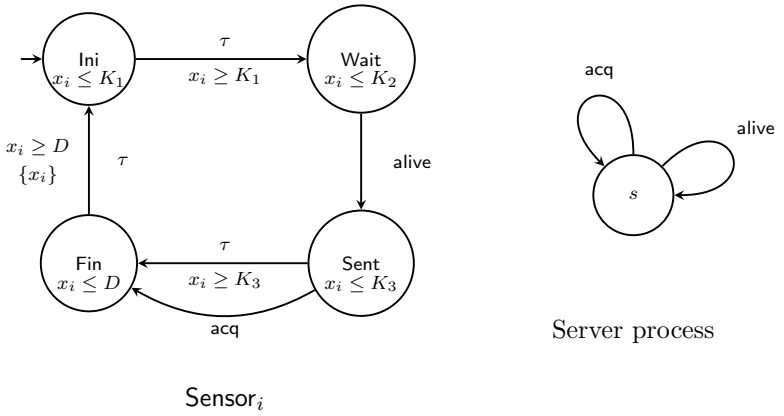
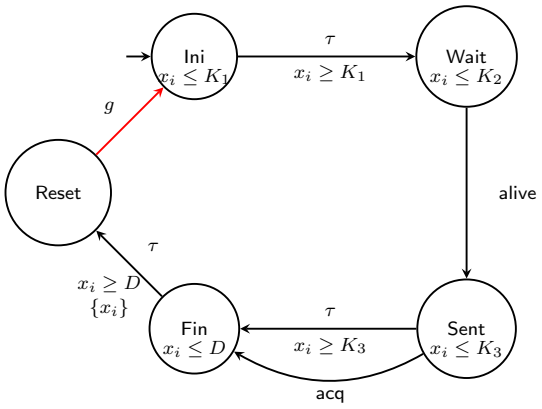


Figure 9.13: Standard Fire-alarm model

In this model, the values of constants K_1, K_2, K_3 differ for each process, and are chosen in such a way that the process Sensor$_i$ executes the action Sent $\rightarrow$ Fin before the process Sensor$_{i+1}$ executes the action Ini $\rightarrow$ Wait, for each $0 \leq i < n$. The value of the constant D depends on the number of processes and is chosen in such a way that all the sensors end their cycle simultaneously, D seconds after the process Sensor$_1$ starts execution. Thus, in each global run of the network, the process Sensor$_i$ reaches the Fin state before the process Sensor$_{i+1}$ starts execution of its cycle. Once each Sensor process reaches the Fin state, each of them executes the action Fin $\rightarrow$ Ini simultaneously, and completes the cycle.

Global-local model. For running our implementation, we consider a global-local variant of this model. Our Fire-alarm global-local model is as depicted in Figure 9.14. Note that, while the action g from Reset to Ini is a global action, all other actions are local actions. In our model, we consider the same values of K_1, K_2 and K_3 for all sensor processes. In this way, we relax the strong restrictions on the order of execution of sensors that existed in the standard model.

Since this modified Fire-alarm model is a global-local system, we can apply our global-local POR on this model. Lemma 9.1 states that the Fire-alarm global-local model has a spread of D. Consequently, we can apply global-local POR with spread set to D on this model.



Sensor_i

Figure 9.14: Fire-alarm global-local model

Lemma 9.1. *Fire-alarm global local model is D-spread bounded, where D is the constant used in the guard on action* Fin $\rightarrow$ Reset *in the model.*

Proof. Consider a run σ of the Fire-alarm global-local model:

$$(q_0, \mathsf{v}_0) \xrightarrow{\sigma_1} (q_1, \mathsf{v}_1) \xrightarrow{g_1} (q_2, \mathsf{v}_2) \xrightarrow{\sigma_2} (q_3, \mathsf{v}_3) \xrightarrow{g_2} (q_4, \mathsf{v}_4) \cdots$$

where σ_i's are sequences of local actions and g_i's are global actions.

Observe that the valuations v_{2i+1} are synchronized valuations, for all $i \geq 0$, as these valuations have to execute global actions. Further, note that in each of these valuations, $\widetilde{x}_1 = \widetilde{x}_2 = \cdots = \widetilde{x}_n$. Now, we examine the maximum difference between reference clocks that can appear in valuations in the sequence of local actions $(q_0, \mathsf{v}_0) \xrightarrow{\sigma_1} (q_1, \mathsf{v}_1)$.

We know that for all valuations in σ_i, we have $\widetilde{x}_1 = \widetilde{x}_2 = \cdots = \widetilde{x}_n$. Further, we know that in all these valuations $t_i - \widetilde{x}_i \leq D$. This implies that $t_i - t_j \leq D$, for all $i, j \in \{1, 2, \cdots, n\}$. □

Table 9.7 presents the result of experiments on instances of the Fire-alarm global-local model with different number of processes.

9.4.2 Type 2: Global-local model where POR does not work

In this section, we focus on the global-local variants of some benchmark models, for which our global-local POR does not perform well. We consider one such model, the Fischer global-local model, and analyse the performance of POR-implementation on this model in detail.

Fischer global-local model

Standard Fischer model. The standard Fischer model has n processes $A_1, \ldots, A_n$ and a common variable id, that is stored in a separate process.

Models	Global ZG			Local SG			POR-zone graph		
(# processes)	visited	stored	time	visited	stored	time	visited	stored	time
2	19	19	0.329s	19	19	0.168s	9	9	0.047s
3	71	71	0.368s	71	71	0.138s	13	13	0.061s
4	271	271	2.269s	271	271	0.114s	17	17	0.299s
5	1,055	1,055	0.102s	1,055	1,055	0.123s	21	21	0.085s
8	65,791	65,791	7.752s	65,791	65,791	29.622s	33	33	1.740s
10	1,049,599	1,049,599	> 1hr	1,049,599	1,049,599	> 1hr	41	41	1.865s

Table 9.7: Experimental results obtained by running prototype TChecker on the Fire-alarm global-local model for different number of processes. Experiments for models of up to 5 processes were run with default values of parameters. For the model with more than 5 processes, experiments were done with values of parameters set to block-size = $100,000,000$ and table-size = $65,536,000$.

Thus, we have a timed automaton for each process, and a timed automaton which stores the shared variable id, as shown in Figure 9.15.

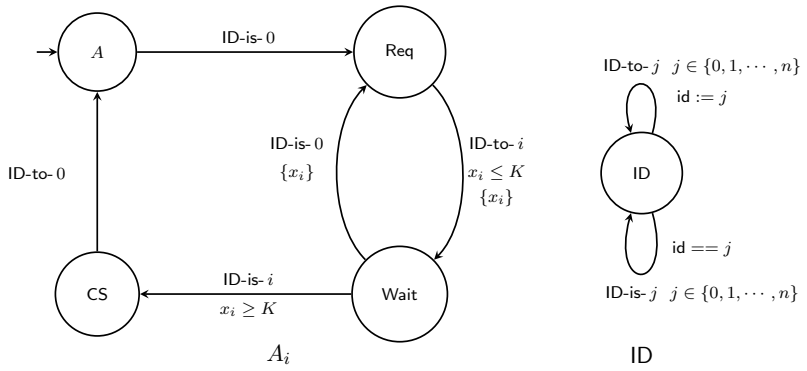


Figure 9.15: Fischer standard model

We now describe the functioning of the Fischer standard model. Each process has four states: A, Req, Wait and CS. Initially it is in the A state. After elapsing an arbitrary amount of time in the A state, the process checks whether the shared variable is currently free (by checking if the value of id is 0). If id is not being accessed by any other process, within k time units, the process declares its intent to access the critical section by setting the value of id to its index, i. After waiting for at least k time units, the process checks that the value of id is still i (makes sure that some other process has not accessed and written into it) and then enters the critical section. If the value of id is no longer i after k time units, it means that some other process is trying to access the critical section and has issued a later request. In this case, the process waits in the state Wait until the critical section is free again and then goes back to the Req state.

Global-local model. Here, we present a global-local variant of the Fischer model where the variable ID permits concurrent reads.

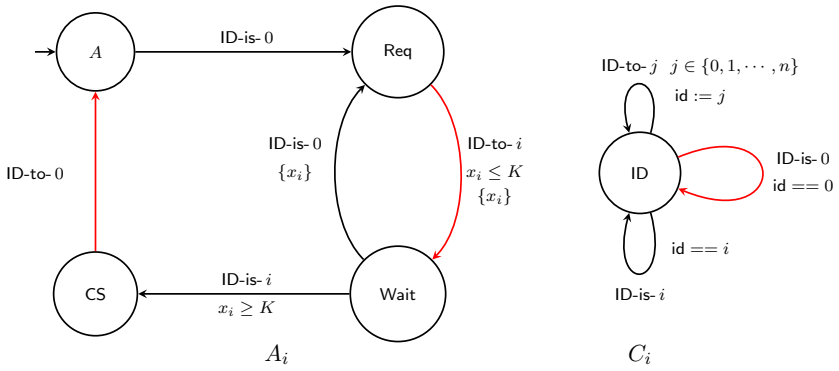


Figure 9.16: Fischer global-local model

To allow this behaviour, we introduce n copies of the variable ID, which we call local cache C_i of the process A_i. Thus, we have a local cache C_i for each process and these cache synchronize on writes. The network is of the form $A_1 \times C_1 \times \cdots \times A_n \times C_n$ where C_i are the caches as given in the Figure 9.16. This system is a global-local system if we consider $(A_1 \times C_1) \times \cdots \times (A_n \times C_n)$ as the network - we consider the product of $A_i \times C_i$ so that it becomes a single automaton. In the resulting system the ID-to- i actions are global actions, while all other actions are local actions.

Note that there are self-loops for each global action on each state of the Fischer product process $(A_i \times C_i)$. As shown in Section 9.3.2, networks in which there are common outgoing global actions from each state are not favourable to our global-local POR technique, as Optimization 1 will not remove any node in the POR-zone graph of this network, as every node has outgoing global actions. Hence, we cannot hope to have any gains in these models.

We generalize this observation further as follows: The processes participating in the models such as Fischer and CSMA/CD have to keep track of a central resource at all times. This central resource could be a shared variable in the case of Fischer, or the status of the shared bus in the case of CSMA/CD. As a consequence, when we model these protocols using global-local networks, we are forced to add global actions reflecting the modification of the status of this common shared resource in each state of all the processes. This is detrimental for the performance of our global-local POR algorithm.

Table 9.8 presents the result of experiments on instances of the Fischer global-local model with different number of processes.

9.4.3 Type 3: Client-server model where POR works

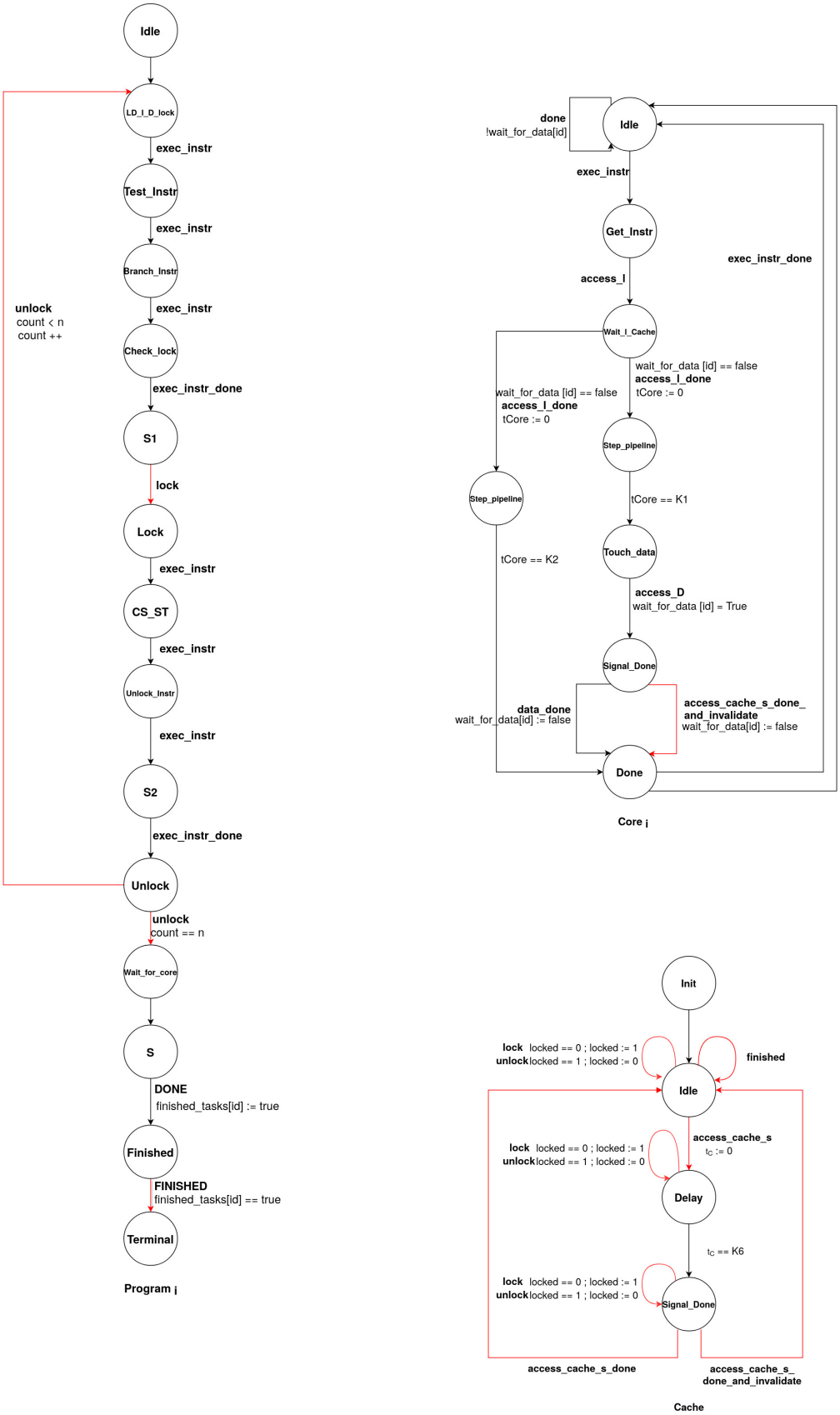


Figure 9.17: WCET Model

Models (# clients)	Global ZG			Local SG			POR-zone graph		
	visited	stored	time	visited	stored	time	visited	stored	time
Fischer gl-2	18	18	0.021s	18	18	0.017s	20	20	0.014
Fischer gl-3	71	65	0.039s	71	65	0.042s	89	89	0.034s
Fischer gl-4	268	220	0.075s	268	220	0.085s	424	424	0.128s
Fischer gl-5	727	977	0.270s	727	977	0.225s	2287	2287	0.526s

Table 9.8: Experimental results obtained by running prototype TChecker on the Fischer global-local model from Figure 9.16 for different number of processes.

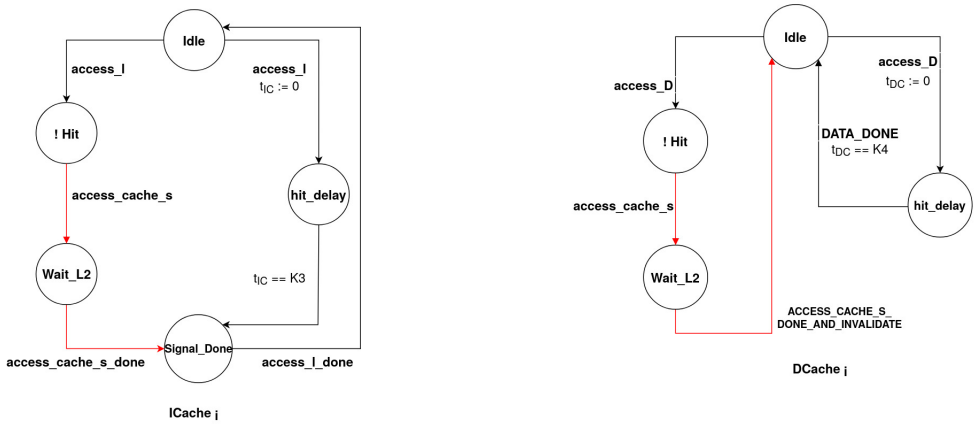


Figure 9.18: WCET Model

WCET model. The model described in this section is inspired from the multicore architecture model introduced by Gustavsson et al. [GELP10], using which they simulate a fictitious shared-memory multicore architecture. Their model consists of k programs operating concurrently with k cores and a main memory. The task of each program involves executing a set of instructions, some of which require accessing and writing into a shared variable. This task is done a fixed number of times before the execution is finished.

The WCET model consists of a server process, called Cache, and four processes for each client i, namely $\mathsf{Program}_i$, Core_i, $\mathsf{I\text{-}Cache}_i$ and $\mathsf{D\text{-}Cache}_i$, given in Figure 9.17 and Figure 9.18. These four processes of each client have mutual synchronizations between them and all the processes have synchronizations with the server.

As already mentioned, the process $\mathsf{Program}_i$ has to execute a set of instructions n times. Some of these instructions require accessing a shared variable, and therefore, mutual exclusion is required for these instructions. This is accomplished by using a lock variable. The status of the lock is stored

in the server in a Boolean variable locked - therefore, the actions involving lock are synchronizations with the server. There are three actions involving the lock variable - the check-locked action which checks that the lock is not free, the lock action which checks that the lock is free and acquires the lock, and the unlock action which checks that the lock is not free and releases the lock. These actions are represented as self-loops in all states in the server.

When a client needs to access a shared variable (this is modelled in the state Check-lock of the process Program), it checks the status of the lock by communicating with the server. If the lock is not free (for instance, if the shared variable is being accessed by some other client), Program goes to the LD-ID-lock state by executing the check-locked action and continues its execution. If the lock is free, Program executes the communication action lock and then proceeds to execute the instructions which required the shared variable. Once the client completes the execution of these instructions, the process Program releases the lock by executing the communication action unlock from the unlock state. This cycle is repeated n times. The model that we consider here, referred to simply as WCET-model, is a variant of the original model. The model is as given in Figure 9.17.

We now describe the processes in some detail.

- $\mathsf{Program}_i$ (Figure 9.17)

  This process models a program that executes a specific set of instructions n times. We model the behaviour of a program by a sequence of exec-instr actions, followed by a exec-instr-done action. Observe that each of these actions are synchronizations with the process Core_i. The process Program also has lock and unlock actions, that are designed for ensuring mutual exclusion, as discussed above. The process Program also has a Finished action which signals the completion of all instructions by the program. Note that the actions lock, unlock and Finished are communication actions, i.e., synchronizations with the server.

- Core_i (Figure 9.17)

  The Core_i process starts its execution by executing the exec-instr action that is a synchronization with $\mathsf{Program}_i$. After executing the exec-instr action, the Core_i process issues request to access the instruction cache and data cache by executing the access-I action with the processes $\mathsf{I\text{-}Cache}_i$ and the access-D action with $\mathsf{D\text{-}Cache}_i$, respectively. The successful completion of the access requests to the instruction cache and the data cache are signalled by the execution of the actions access-I-done (jointly with $\mathsf{I\text{-}Cache}_i$) and access-D-done (jointly with $\mathsf{D\text{-}Cache}_i$), respectively. Upon completion of the cache access, Core_i signals it to $\mathsf{Program}_i$ by executing the exec-instr-done action. The Core_i process also executes the action done jointly with $\mathsf{Program}_i$, the purpose of which is to signal to the program that the core is not in the

middle of its execution. Observe that Core_i has "wait states" which requires the elapse of a certain amount of time in them.

- $\mathsf{I\text{-}Cache}_i$ (Figure 9.18)

  The I-Cache process is meant to simulate the working of the instruction cache of the architecture. Once the Core issues a request to access the instruction cache by executing the access-I action, if there is a Cache hit, the I-Cache executes the access-I-done action after waiting for at least K_3 time units. Otherwise, I-Cache executes the action access-cache-s, which indicates a request to access the secondary cache. Note that this is a communication action with the server process.

- $\mathsf{D\text{-}Cache}_i$ (Figure 9.18)

  Just as the I-Cache process simulates the working of the instruction cache, the D-Cache process simulates the working of the data cache of the architecture. Once the Core issues a request to access the data cache by executing the access-D action, if there is a Cache hit, the D-Cache executes the Data-done action after at least K_4 time units. Otherwise, D-Cache executes the action access-cache-s, which indicates a request to access the secondary cache and is a communication action with the server process.

The server simulates the secondary cache and stores a Boolean variable called "locked", which is used to ensure mutual exclusion, whenever required.

We now demonstrate that this model is an extended client-server model (see Section 8.4.2). For ease of reference, we denote by WCET_i the set $\{\mathsf{Program}_i, \mathsf{Core}_i, \mathsf{I\text{-}Cache}_i, \mathsf{D\text{-}Cache}_i\}$. The synchronizations in the model are of the following formulated

- communication actions involving the server and processes from the set WCET_i, for some $i \in \{1, \ldots, n\}$.
- synchronizations involving processes from the set WCET_i.

Observe that there are no synchronizations between a process from WCET_i and WCET_j, where $i \neq j$. Further, the only timing constraints in the WCET model are of the form of wait states. It may be observed that the WCET model is an example of a client-server network with wait states, that are considered in Lemma 7.6. Directly from the lemma, it follows that the WCET model is spread bounded with a bound of $N \cdot W$, where N is the number of wait states of a process that can appear in a run of $\mathcal{N}$, and W is an upper bound on the wait time associated to a wait state in $\mathcal{N}$. We apply our client-server POR to the WCET model, with the parameter spread set to 1000, which is greater than $N \cdot W$ in this case.

Table 9.9 presents the number of visited nodes and stored nodes in the global zone graph, local sync graph and the $\mathsf{POR\text{-}LZG}_r$ of instances of the

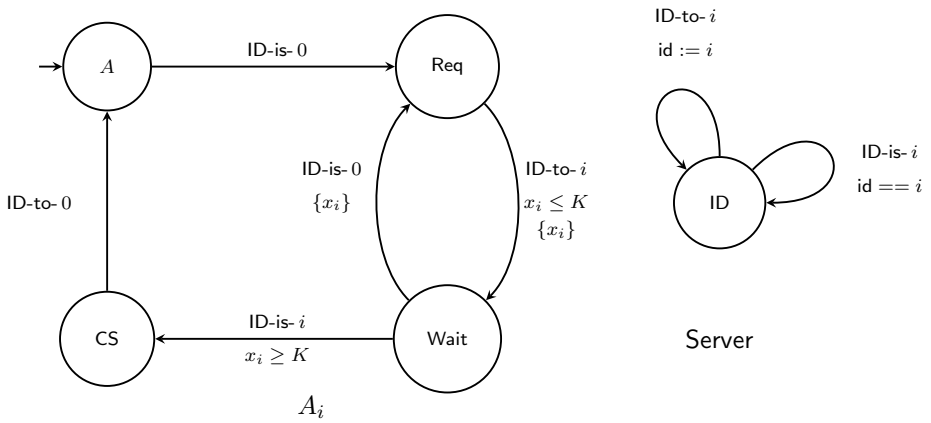
WCET model for different number of clients, and the respective runtime of the procedures. Note that the results in the Table 9.9 are for the WCET model with the value of n set to 1.

Models	Global ZG			Local SG			POR-zone graph		
(# clients)	visited	stored	time	visited	stored	time	visited	stored	time
1	138	138	2.260s	138	138	2.339s	113	101	2.287s
2	9,379	9,379	1m5.354s	8,803	8,803	53.081s	6260	4520	1m5.344s
3	647,338	647,338	95m19.460s	524,650	524,650	81m31.305s	168,463	114,319	11m4.878s
4	47,084,877	47,084,877	> 1,027m29s	29,648,493	29,648,493	> 472m31s	3,608,470	2,204,254	200m49.445s

Table 9.9: Experimental results for the WCET model given in Figure 9.17 for different number of clients. Experiments were done with values of parameters set to block-size $= 100,000,000$ and table-size $= 65,536,000$.

9.4.4 Type 4: Client-server model where POR does not work

Fischer-client-server. Recall the standard Fischer model that we introduced in Section 9.4.2. In this model, we have a timed automaton for each process, and a timed automaton which stores the shared variable id.



Standard Fischer model

Figure 9.19: Fischer client-server model

Observe that this model falls under the category of the client-server models. However, each action of a client is synchronized with the server, i.e., each action in this model is a communication action. This means that there is no concurrency in this model, and consequently no hope of getting any reduction with our POR-reachability algorithm.

9.4.5 Type 5: Models that are spread-unbounded

Dining philosophers model. In this section, we consider a timed version of the classical Dining Philosophers protocol, inspired from a model presented in Lugiez et al. [LNZ05]. The classical model involves a certain number of philosophers who are seated around a table with a fork placed between each of them, and the philosophers are required to have dinner respecting certain conditions. A philosopher can only eat when she has forks placed to her left and right. Further, each fork can only be held by one philosopher at a time.

Our Dining Philosophers model consists of n processes modelling philosophers and n identical processes modelling forks. We have a timed automaton for each process as depicted in Figure 9.20. In our model, we have a timeout to release the acquired fork if the other fork cannot be obtained within a fixed time, in order to avoid deadlocks.

The Fork automaton has two states - free and taken. If the automaton is in free state, it signifies that the fork is free to be acquired, whereas if it is in taken state, it means that the fork is currently being held by a philosopher. Whenever a philosopher wants to acquire a fork, the process modelling that philosopher synchronizes with the Fork automaton to execute the take action which changes the state from free to taken. Similarly, when a fork is released by a philosopher, the process modelling it synchronizes with the respective fork automaton to do the rel action, and changes its state from taken to free.

The timed automaton modelling Philosopher has four states -Idle, Acq, Eat and Release. The automaton is initially in the state Idle. Recall that $\mathsf{Philosopher}_i$ uses Fork_{i-1} and Fork_i to eat. After elapsing an arbitrary amount of time in the Idle state, $\mathsf{Philosopher}_i$ checks whether Fork_{i-1} is currently free; if it is free, it is acquired. Within K_1 time units of acquiring Fork_{i-1}, $\mathsf{Philosopher}_i$ tries to acquire Fork_i. If it is not possible to acquire Fork_i within K_1 time units, Fork_{i-1} is released and $\mathsf{Philosopher}_i$ goes back to Idle state. On the other hand, if she manages to acquire Fork_i, the philosopher proceeds to Eat state, where she spends K_2 time units (the duration of the dinner). Exactly after spending K_2 time units in the Eat state, the Philosopher releases both forks in zero time and goes to Idle state.

Next, we will show that the Dining Philosophers model given in Figure 9.20 is not spread bounded.

Lemma 9.2. *The Dining Philosophers model is not spread bounded.*

Proof. Consider the instance of the Dining Philosophers model with 4 philosophers and 4 forks. Recall that $\mathsf{Philosopher}_1$ needs Fork_1 and Fork_4 to eat, $\mathsf{Philosopher}_3$ needs Fork_2 and Fork_3, and so on. Further, assume that the duration of dinner is d time units.

Consider the situation where $\mathsf{Philosopher}_1$ has acquired Fork_4 - the system is now in the state $(\mathsf{Acq}, \mathsf{Idle}, \mathsf{Idle}, \mathsf{Idle})$ for the philosophers. Now, suppose $\mathsf{Philosopher}_3$ acquires Fork_2 and Fork_3. At this point, the state of philosophers

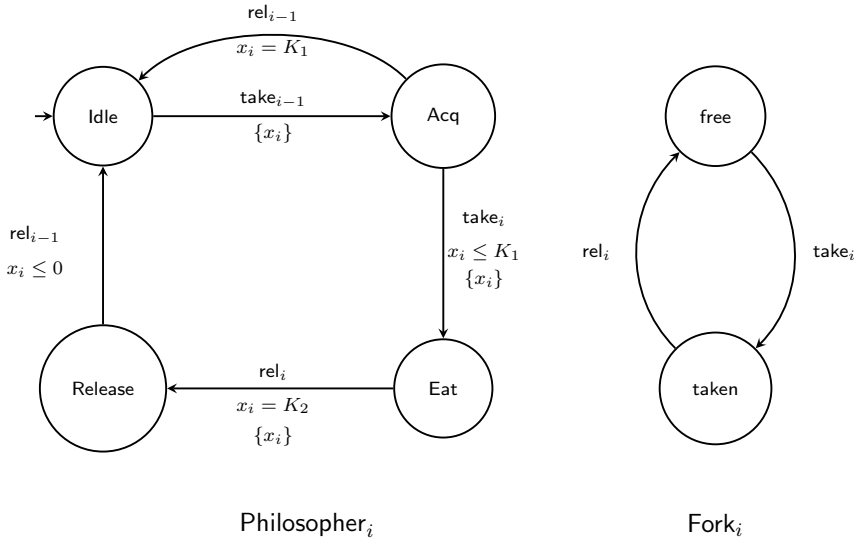


Figure 9.20: Dining Philosophers model

is: (Acq, Idle, Eat, Idle). From this state, $\mathsf{Philosopher}_3$ can execute the loop: Eat $\rightarrow$ Release $\rightarrow$ Idle $\rightarrow$ Acq an arbitrary number of times (say k times), thereby increasing the difference of local time of the automata $\mathsf{Philosopher}_1$ and $\mathsf{Philosopher}_3$ by $K \cdot d$.

Note that the action take_1 of $\mathsf{Philosopher}_1$ (from the state Acq) has an upper bound guard associated to it. So, in a local run which executes this action, $\mathsf{Philosopher}_1$ cannot elapse time arbitrarily and keep up with the local time of $\mathsf{Philosopher}_3$. As a result the reference clocks of $\mathsf{Philosopher}_1$ and $\mathsf{Philosopher}_3$ can diverge arbitrarily from each other.

Next, we show that the network can reach a synchronized valuation from the aforementioned valuation of large spread. This is accomplished by $\mathsf{Philosopher}_1$ executing the loop: Eat $\rightarrow$ Release $\rightarrow$ Idle $\rightarrow$ Acq and catching up with the local time of $\mathsf{Philosopher}_3$ - after which the processes can do a transitive synchronization, where $\mathsf{Philosopher}_1$ synchronizes with Fork_1, then Fork_1 synchronizes with $\mathsf{Philosopher}_2$, followed by synchronization between Fork_2 and $\mathsf{Philosopher}_3$.

The run given above is a local-time run, which has no equivalent k-spread run. This implies that the system is not k-spread, for any value of k. In other words, it is not spread bounded. □

Chapter 10

Conclusion

The key contributions of this thesis are a systematic study of state-space explosion in the verification of networks of timed automata caused due to interleaving of the components and solutions for alleviating the effects of this explosion. We believe that these are important steps towards obtaining scalable verification procedures for various classes of timed automata. In this chapter, we summarize the work presented in this thesis, and identify some directions for future research.

10.1. Summary of our contributions

We provide two different solutions for the state-space explosion problem in verification of networks of timed automata. The first one is an algorithm that tests the reachability of networks of timed automata based on the exploration of the local zone graph of the network. The second is a framework for partial order reduction methods for networks of timed automata and POR algorithms for some classes of timed automata.

In order to design our algorithm, we revisit the local time semantics of timed automata and local zone graphs [BJLY98]. We discover a very useful fact which says that local zones compute aggregated zones [SBM06], where an aggregated zone is a union of all the global zones obtained by equivalent executions. We use this fact as a theoretical foundation to design an algorithm for constructing local zone graphs using subsumption on aggregated zones. Furthermore, we show that the existing techniques that use subsumption operations on local zones [BJLY98, Min99a, Min99b] do not work. We propose a new subsumption for local zone graphs based on standard abstractions for timed automata, where the subsumption is applied on synchronized zones. The restriction to synchronized zones is crucial as standard abstractions cannot handle multiple reference clocks. Our algorithm is the first efficient implementation of local zone graphs and aggregated zones.

Experimental results show an order of magnitude gain with respect to state of the art algorithms on several standard examples.

We remark that a natural progression for this work is to apply standard partial order techniques to the aforementioned algorithm. Unfortunately, we do not have an effective way to compute the independence relation between actions in the finite local zone graph that we construct. Consequently, we cannot apply a partial order reduction method on this transition system.

Next, we propose a region equivalence for local valuations and show that this region equivalence is a simulation relation for local valuations. Furthermore, building on this region equivalence, we propose a new subsumption relation, which we call $\mathfrak{a}^*_M$ subsumption. However, the transition system obtained by applying the $\mathfrak{a}^*_M$ subsumption to local zone graphs is not finite in general. We identify that the fundamental difficulty in designing a finite subsumption for local zone graphs is the arbitrary divergence between the reference clocks of different processes of the network. Equipped with this understanding, we define the spread of a network of timed automata to be the maximum divergence between reference clocks across processes and restrict our attention to networks that are spread-bounded.

We modify the $\mathfrak{a}^*_M$ subsumption to design a new subsumption relation, called $\mathfrak{a}^D_M$ subsumption, that is parameterized by a constant D. By applying the $\mathfrak{a}^D_M$ subsumption while exploring the local zone graph of a network, one obtains a finite transition system, denoted $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}$. We show that if the spread of a network is bounded by D, then the $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}$ of the network is sound and complete with respect to reachability. Further, we give a condition that computes an approximation of the independence relation of the $\mathsf{LZG}^{\mathsf{D}}_{\mathsf{M}}$ of a D-spread-bounded network. We identify two classes of networks of timed automata, referred to as global-local systems and client-server systems, respectively, and propose partial order reduction methods for spread-bounded networks belonging to these categories. We also provide an evaluation of a prototype of the implementation of these methods on some examples using the tool TChecker [HP19].

10.2. Directions for future research

The main directions for future research are to develop better partial order reduction methods for networks of timed automata. We have proposed a framework for applying partial order reduction to networks of timed automata, and proposed POR-algorithms for some classes of timed automata. Refining our algorithm to optimize it further, and expanding its applicability to more classes is an important direction for immediate future research. It is also desirable to carry out further experimental investigations to determine the effectiveness of our POR-reachability algorithms.

The scope of our techniques is restricted to networks of simple timed automata. In particular, classes of timed automata with more behavior, such as timed automata with diagonal constraints [AD94], updatable timed automata [BDFP04], priced timed automata [BFLM11] and probabilistic timed automata [KNSS02] fall outside the scope of our technique. Generalizing our technique to allow its application to these more general models is a broader and more ambitious research direction.

We list some other open problems below.

Computing the spread of a given network of timed automata. The POR-reachability procedure that we introduce in Chapter 8 can only be applied to networks of timed automata that are spread-bounded. Consequently, it is crucial to have ways to check if a network is spread bounded and compute an upper bound on the spread of the network. In Chapter 7, we propose some efficiently checkable syntactic conditions on a network that imply bounds on its spread. Identifying more such efficiently checkable sufficient conditions is an immediate open question of interest. A more challenging question would be to obtain a characterization of networks of timed automata of a specified spread.

Further, it would be interesting to design an efficient procedure that takes a network of timed automata as input and outputs the spread of the network. We believe that such a procedure would significantly enhance the applicability of our POR-reachability procedure.

Analogue of $\mathfrak{a}_{\preccurlyeq LU}$ subsumption. The $\mathfrak{a}^D_M$ subsumption for local zones proposed in Chapter 6 is analogous to the standard $\mathfrak{a}_M$ subsumption for standard zones. In the setting of standard zone graphs, it is known that the $\mathfrak{a}_{\preccurlyeq LU}$ subsumption produces zone graphs with fewer nodes than the $\mathfrak{a}_M$ subsumption. So, defining an analogue of the $\mathfrak{a}_{\preccurlyeq LU}$ subsumption for local zones has the potential to make the POR-reachability algorithm more efficient.

Optimizations to POR-reachability algorithm. We have proposed separate POR-reachability procedures for global-local systems. Further investigation of the global-local and client-server POR algorithms and designing optimizations to make these procedures more efficient is an interesting line of work.

Semi-commutation in global-zone graphs. In this thesis, we work predominantly in the setting of local-time semantics. It may also be possible to come up with partial order reduction techniques in the standard semantics. In the standard setting, this would involve looking at sequences and checking if a given sequence is *semi-commutable*. This direction has already been

investigated in Dams et al. [DGKK98] and Hansen et al.[HLL$^+$14]. We think further exploration along these lines might prove useful.

Adapting POR-reachability algorithm for specific models. We have proposed a POR-reachability procedure for the general class of network of timed automata. In various subclasses of networks of timed automata such as timed automata with only one clock, timed automata with urgent behavior etc., it may be possible to exploit the specific properties of these subclasses to optimize the performance of out POR-reachability procedure further. Adapting the algorithm to specific subclasses of networks of timed automata is an essential direction of study.

Complexity of the D-spread-boundedness problem. In Chapter 7, we show that the D-spread-boundedness problem is PSPACE-hard. However, we have not given an upper bound for the complexity of the D-spread-boundedness problem. proving a concrete upper bound for the complexity of the D-spread-boundedness problem is an interesting open question.

Bibliography

[AAJS17] Parosh Aziz Abdulla, Stavros Aronis, Bengt Jonsson, and Konstantinos Sagonas. Source sets: A foundation for optimal dynamic partial order reduction. *J. ACM*, 64(4):25:1–25:49, 2017.

[AD90] Rajeev Alur and David L. Dill. Automata for modeling real-time systems. In *ICALP*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. Springer, 1990.

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[BBFL03] Gerd Behrmann, Patricia Bouyer, Emmanuel Fleury, and Kim Guldstrand Larsen. Static guard analysis in timed automata verification. In *TACAS*, volume 2619 of *Lecture Notes in Computer Science*, pages 254–277. Springer, 2003.

[BBLP06] Gerd Behrmann, Patricia Bouyer, Kim G. Larsen, and Radek Pelánek. Lower and upper bounds in zone-based abstractions of timed automata. *International Journal on Software Tools for Technology Transfer*, 8(3):204–215, 2006.

[BCC+99] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Masahiro Fujita, and Yunshan Zhu. Symbolic model checking using SAT procedures instead of BDDs. In *DAC*, pages 317–320. ACM Press, 1999.

[BDFP04] Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury, and Antoine Petit. Updatable timed automata. *Theor. Comput. Sci.*, 321(2-3):291–345, 2004.

[BDL+06] Gerd Behrmann, Alexandre David, Kim Guldstrand Larsen, John Hakansson, Paul Pettersson, Wang Yi, and Martijn Hendriks. UPPAAL 4.0. In *QEST*, pages 125–126. IEEE Computer Society, 2006.

[BDM+98] Marius Bozga, Conrado Daws, Oded Maler, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. Kronos: A model-checking

tool for real-time systems. In *CAV*, volume 1427 of *Lecture Notes in Computer Science*, pages 546–550. Springer, 1998.

[Ben02] Johan Bengtsson. *Clocks, DBMs and States in Timed Systems*. PhD thesis, Department of Information Technology, Uppsala University, Uppsala, Sweden, 2002.

[BFLM11] Patricia Bouyer, Uli Fahrenberg, Kim G. Larsen, and Nicolas Markey. Quantitative analysis of real-time systems using priced timed automata. *Commun. ACM*, 54(9):78–87, 2011.

[BHR06] Patricia Bouyer, Serge Haddad, and Pierre-Alain Reynier. Timed unfoldings for networks of timed automata. In *ATVA*, volume 4218 of *Lecture Notes in Computer Science*, pages 292–306. Springer, 2006.

[BJL+18] Frederik M. Bønneland, Peter Gjøl Jensen, Kim Guldstrand Larsen, Marco Muñiz, and Jirí Srba. Start pruning when time gets urgent: Partial order reduction for timed systems. In *CAV*, volume 10981 of *Lecture Notes in Computer Science*, pages 527–546. Springer, 2018.

[BJLY98] Johan Bengtsson, Bengt Jonsson, Johan Lilius, and Wang Yi. Partial order reductions for timed systems. In *CONCUR*, volume 1466 of *Lecture Notes in Computer Science*, pages 485–500, 1998.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[BM83] Bernard Berthomieu and Miguel Menasche. An enumerative approach for analyzing time petri nets. In *IFIP Congress*, pages 41–46. North-Holland/IFIP, 1983.

[Bou09] Patricia Bouyer. *From Qualitative to Quantitative Analysis of Timed Systems*. Mémoire d'habilitation, Université Paris 7, Paris, France, 2009.

[BPDG98] Béatrice Bérard, Antoine Petit, Volker Diekert, and Paul Gastin. Characterization of the expressive power of silent transitions in timed automata. *Fundam. Informaticae*, 36(2-3):145–182, 1998.

[BY03] Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In *ACPN 2003*, volume 3098 of *Lecture Notes in Computer Science*, pages 87–124. Springer, 2003.

[CCJ06] Franck Cassez, Thomas Chatain, and Claude Jard. Symbolic unfoldings for networks of timed automata. In *ATVA*, volume 4218

of *Lecture Notes in Computer Science*, pages 307–321. Springer, 2006.

[CG18] Sagar Chaki and Arie Gurfinkel. BDD-based symbolic model checking. In *Handbook of Model Checking*, pages 219–245. Springer, 2018.

[CGJ+03] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM*, 50(5):752–794, 2003.

[CGMP99] Edmund M. Clarke, Orna Grumberg, Marius Minea, and Doron A. Peled. State space reduction using partial order techniques. *Int. J. Softw. Tools Technol. Transf.*, 2(3):279–287, 1999.

[CGP01] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model checking*. MIT Press, 2001.

[CHVB18] Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors. *Handbook of Model Checking*. Springer, 2018.

[CJ06] Thomas Chatain and Claude Jard. Complete finite prefixes of symbolic unfoldings of safe time petri nets. In *ICATPN*, volume 4024 of *Lecture Notes in Computer Science*, pages 125–145. Springer, 2006.

[CJ13] Thomas Chatain and Claude Jard. Back in time petri nets. In *FORMATS*, volume 8053 of *Lecture Notes in Computer Science*, pages 91–105. Springer, 2013.

[CKNZ11] Edmund M. Clarke, William Klieber, Milos Novácek, and Paolo Zuliani. Model checking and the state explosion problem. In *LASER Summer School*, volume 7682 of *Lecture Notes in Computer Science*, pages 1–30. Springer, 2011.

[DGKK98] Dennis Dams, Rob Gerth, Bart Knaack, and Ruurd Kuiper. Partial-order reduction techniques for real-time model checking. *Formal Asp. Comput.*, 10(5-6):469–482, 1998.

[Dil89] David L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*, pages 197–212. Springer, 1989.

[DOTY95] Conrado Daws, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. The tool KRONOS. In *Hybrid Systems*, volume 1066 of *Lecture Notes in Computer Science*, pages 208–219. Springer, 1995.

[DT98] Conrado Daws and Stavros Tripakis. Model checking of real-time reachability properties using abstractions. In *TACAS*, volume 1384 of *Lecture Notes in Computer Science*, pages 313–329. Springer, 1998.

[ERV02] Javier Esparza, Stefan Römer, and Walter Vogler. An improvement of McMillan's unfolding algorithm. *Formal Methods Syst. Des.*, 20(3):285–310, 2002.

[FG05] Cormac Flanagan and Patrice Godefroid. Dynamic partial-order reduction for model checking software. In *POPL*, pages 110–121. ACM, 2005.

[GELP10] Andreas Gustavsson, Andreas Ermedahl, Björn Lisper, and Paul Pettersson. Towards WCET analysis of multicore architectures using UPPAAL. In *WCET*, volume 15 of *OASICS*, pages 101–112. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2010.

[GHSW19] R. Govind, Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. Revisiting local time semantics for networks of timed automata. In *CONCUR*, volume 140 of *LIPIcs*, pages 16:1–16:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[GMS19] Paul Gastin, Sayan Mukherjee, and B. Srivathsan. Fast algorithms for handling diagonal constraints in timed automata. *CoRR*, abs/1904.08590, 2019.

[God90] Patrice Godefroid. Using partial orders to improve automatic verification methods. In *CAV*, volume 531 of *Lecture Notes in Computer Science*, pages 176–185. Springer, 1990.

[God96] Patrice Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems - An Approach to the State-Explosion Problem*, volume 1032 of *Lecture Notes in Computer Science*. Springer, 1996.

[GPS96] Patrice Godefroid, Doron A. Peled, and Mark G. Staskauskas. Using partial-order methods in the formal validation of industrial concurrent programs. *IEEE Trans. Software Eng.*, 22(7):496–507, 1996.

[HHW97] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. HYTECH: A model checker for hybrid systems. *Int. J. Softw. Tools Technol. Transf.*, 1(1-2):110–122, 1997.

[HKSW11] Frédéric Herbreteau, Dileep Kini, B. Srivathsan, and Igor Walukiewicz. Using non-convex approximations for efficient analysis of timed automata. In *FSTTCS*, volume 13 of *LIPIcs*, pages 78–89. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011.

[HLL+14] Henri Hansen, Shang-Wei Lin, Yang Liu, Truong Khanh Nguyen, and Jun Sun. Diamonds are a girl's best friend: Partial order reduction for timed automata with abstractions. In *CAV*, volume 8559 of *Lecture Notes in Computer Science*, pages 391–406. Springer, 2014.

[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.

[HP94] Gerard J. Holzmann and Doron A. Peled. An improvement in formal verification. In *FORTE*, volume 6 of *IFIP Conference Proceedings*, pages 197–211. Chapman & Hall, 1994.

[HP19] F. Herbreteau and G. Point. TChecker. `https://github.com/fredher/tchecker`, v0.2 - April 2019.

[HSW12] Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. Better abstractions for timed automata. In *LICS*, pages 375–384. IEEE Computer Society, 2012.

[KLM+98] Robert P. Kurshan, Vladimir Levin, Marius Minea, Doron A. Peled, and Hüsnü Yenigün. Static partial order reduction. In *TACAS*, volume 1384 of *Lecture Notes in Computer Science*, pages 345–357. Springer, 1998.

[KLM+15] Gijs Kant, Alfons Laarman, Jeroen Meijer, Jaco van de Pol, Stefan Blom, and Tom van Dijk. LTSmin: High-performance language-independent model checking. In *TACAS*, volume 9035 of *Lecture Notes in Computer Science*, pages 692–707. Springer, 2015.

[KNSS02] Marta Z. Kwiatkowska, Gethin Norman, Roberto Segala, and Jeremy Sproston. Automatic verification of real-time systems with discrete probability distributions. *Theor. Comput. Sci.*, 282(1):101–150, 2002.

[KP92] Shmuel Katz and Doron A. Peled. Defining conditional independence using collapses. *Theor. Comput. Sci.*, 101(2):337–359, 1992.

[Lam77] Leslie Lamport. Proving the correctness of multiprocess programs. *IEEE Trans. Software Eng.*, 3(2):125–143, 1977.

[LL98] François Laroussinie and Kim Guldstrand Larsen. CMC: A tool for compositional model-checking of real-time systems. In *FORTE*, volume 135 of *IFIP Conference Proceedings*, pages 439–456. Kluwer, 1998.

[LNZ05] Denis Lugiez, Peter Niebert, and Sarah Zennou. A partial order semantics approach to the clock explosion problem of timed automata. *Theor. Comput. Sci.*, 345(1):27–59, 2005.

[LPY97] Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. *STTT*, 1(1-2):134–152, 1997.

[LS00] François Laroussinie and Philippe Schnoebelen. The state explosion problem from trace to bisimulation equivalence. In *FoSSaCS*, volume 1784 of *Lecture Notes in Computer Science*, pages 192–207. Springer, 2000.

[Maz86] Antoni W. Mazurkiewicz. Trace theory. In *Advances in Petri Nets*, volume 255 of *Lecture Notes in Computer Science*, pages 279–324. Springer, 1986.

[McM95] Kenneth L. McMillan. Trace theoretic verification of asynchronous circuits using unfoldings. In *CAV*, volume 939 of *Lecture Notes in Computer Science*, pages 180–195. Springer, 1995.

[Min99a] Marius Minea. Partial order reduction for model checking of timed automata. In *CONCUR*, volume 1664 of *Lecture Notes in Computer Science*, pages 431–446. Springer, 1999.

[Min99b] Marius Minea. *Partial Order Reduction for Verification of Timed Systems*. PhD thesis, School of Computer Science, Carnegie Mellon University Pittsburgh, PA 15213, 1999.

[MPS11] Georges Morbé, Florian Pigorsch, and Christoph Scholl. Fully symbolic model checking for timed automata. In *CAV*, volume 6806 of *Lecture Notes in Computer Science*, pages 616–632. Springer, 2011.

[MWP12] Marco Muñiz, Bernd Westphal, and Andreas Podelski. Timed automata with disjoint activity. In *FORMATS*, volume 7595 of *Lecture Notes in Computer Science*, pages 188–203. Springer, 2012.

[OG76] Susan S. Owicki and David Gries. Verifying properties of parallel programs: An axiomatic approach. *Commun. ACM*, 19(5):279–285, 1976.

[Pel93] Doron A. Peled. All from one, one for all: on model checking using representatives. In *CAV*, volume 697 of *Lecture Notes in Computer Science*, pages 409–423. Springer, 1993.

[Pel96] Doron A. Peled. Combining partial order reductions with on-the-fly model-checking. *Formal Methods Syst. Des.*, 8(1):39–64, 1996.

[SBM06] Ramzi Ben Salah, Marius Bozga, and Oded Maler. On interleaving in timed automata. In *CONCUR*, volume 4137 of *Lecture Notes in Computer Science*, pages 465–476. Springer, 2006.

[Sho81] Robert E. Shostak. Deciding linear inequalities by computing loop residues. *J. ACM*, 28(4):769–779, 1981.

[SLDP09] Jun Sun, Yang Liu, Jin Song Dong, and Jun Pang. PAT: Towards flexible verification under fairness. volume 5643 of *Lecture Notes in Computer Science*, pages 709–714. Springer, 2009.

[SRDK17] Marcelo Sousa, César Rodríguez, Vijay D'Silva, and Daniel Kroening. Abstract interpretation with unfoldings. In *CAV*, pages 197–216, 2017.

[Sri12] Balaguru Srivathsan. *Abstractions for timed automata.* Phd thesis, Université de Bordeaux, 2012.

[TAKB96] Serdar Tasiran, Rajeev Alur, Robert P. Kurshan, and Robert K. Brayton. Verifying abstractions of timed systems. In *CONCUR*, volume 1119 of *Lecture Notes in Computer Science*, pages 546–562. Springer, 1996.

[THV+17] Tamás Tóth, Ákos Hajdu, András Vörös, Zoltán Micskei, and István Majzik. Theta: a framework for abstraction refinement-based model checking. In Daryl Stewart and Georg Weissenbacher, editors, *Proceedings of the 17th Conference on Formal Methods in Computer-Aided Design*, pages 176–179, 2017.

[TY01] Stavros Tripakis and Sergio Yovine. Analysis of timed systems using time-abstracting bisimulations. *Formal Methods Syst. Des.*, 18(1):25–68, 2001.

[Val89] Antti Valmari. Stubborn sets for reduced state space generation. In *Applications and Theory of Petri Nets*, volume 483 of *Lecture Notes in Computer Science*, pages 491–515. Springer, 1989.

[Val90] Antti Valmari. A stubborn attack on state explosion. In *CAV*, volume 531 of *Lecture Notes in Computer Science*, pages 156–165. Springer, 1990.

[YW06] Fang Yu and Bow-Yaw Wang. SAT-based model checking for region automata. *Int. J. Found. Comput. Sci.*, 17(4):775–796, 2006.

Appendix A

Models with state invariants

In this section, we show that the scope of our techniques can be extended to the more generic setting of networks of timed automata with state invariants. Introduced by Henzinger et al. [HNSY94], a *timed automaton with state invariants* is one where each state of the automaton is associated to a set of constraints of the form $x \sim c$, where $x \in X$ is a clock of the automaton, c is a non-negative integer and $\sim \in \{\leq, <, =, \geq, >\}$. We show that we can convert any network of timed automata with state invariants to a network of timed automata without state invariants, such that the networks are equivalent w.r.t. local time reachability.

A.1. Definitions

In this section, we will present some basic definitions for networks of timed automata with state invariants. Recall that $\Phi(X)$ is the set of clock constraints over the set of clocks X, where each clock constraint is of the form $x \sim c$, where $x \in X$, $c \in \mathbb{N}$, and $\sim \in \{<, \leq, =, \geq, >\}$. We use $\prec$ to denote either $<$ or $\leq$, and $\succ$ to denote either $>$ or $\geq$.

Definition A.1 (Timed automaton with state invariants)**.** A timed automaton with state invariants, A, is given by a tuple $(Q, \Sigma, X, T, q^0, F, \mathsf{Inv})$, where Q is a finite set of states, X is a (finite) set of clocks, Σ is a finite alphabet of actions, $q^0 \in Q$ is the initial state, $F \subseteq Q$ is the set of accepting states. T is a finite set of transitions of the form (q, g, a, R, q') where $q \in Q$ is the source state, $q' \in Q$ is the target state, $g \in \Phi(X)$ is a clock constraint over X, $R \subseteq X$ is the set of clocks reset, and $a \in \Sigma$. Finally, $\mathsf{Inv} : Q \mapsto 2^{\Phi(X)}$ associates an invariant from Φ_X to states from Q.

Remark. *Note that we do not consider invariants of the form $x - y \sim c$. While invariants of the form $x - y \sim c$ can be eliminated using the techniques discussed in this section, the resultant network would contain*

diagonal constraints (see remark in page 23), which is not in the purview of this thesis.

We work with networks $\mathcal{N} = \{A_1, A_2, \cdots, A_n\}$, where each process $A_i = (Q_i, \Sigma_i, X_i, T_i, q_i^0, F_i, \mathsf{Inv}_i)$ is a timed automaton with state invariants. We also do not allow shared clocks in our networks, i.e., $X_i \cap X_j = \emptyset$, for all $i, j \in \mathsf{Proc}$.

We now define local time semantics for networks of timed automata with invariants. Recall that in local time semantics, we have the notion of a local delay. A local delay $\delta \in \mathbb{R}_{\geq 0}$ in process $p \in \mathit{Proc}$ is a step $(q, \mathsf{v}) \xrightarrow{p,\delta} (q, \mathsf{v} +_p \delta)$. For a sequence of local delays $\Delta = (p_1, \delta_1) \ldots (p_n, \delta_n)$ we will write $(q, \mathsf{v}) \xrightarrow{\Delta} (q, \mathsf{v}')$ to mean $(q, \mathsf{v}) \xrightarrow{p_1,\delta_1} (q, \mathsf{v}_1) \xrightarrow{p_2,\delta_2} \cdots \xrightarrow{(p_n,\delta_n)} (q, \mathsf{v}')$. We write $q(p)$ to denote the location of process A_p in the state q of $\mathcal{N}$. In other words, if the state q is a tuple of the form $(q_1, \cdots, q_n)$, then $q(p) = q_p$.

There are two kinds of local steps in a network $\mathcal{N}$: *local delay*, and *local action*.

- **Local delay**: Let $\Delta = (p_1, \delta_1) \ldots (p_n, \delta_n)$. $(q, \mathsf{v}) \xrightarrow{\Delta} (q, \mathsf{v}')$ if for each process $p \in \mathsf{Proc}$, $\mathsf{v}'(t_p) = \mathsf{v}(t_p) + \delta_p$ and for every clock $x \in X \setminus T$, $\mathsf{v}'(x) = \mathsf{v}(x)$, and $\mathsf{v}' \models \bigwedge_{p \in \mathsf{Proc}} \mathsf{Inv}_p(q(p))$;

- **Local action**: For an action b, we have a step $(q, \mathsf{v}) \xrightarrow{b} (q', \mathsf{v}')$ if for each $p \in \mathsf{dom}(b)$ the unique b transition of the process A_p is $T_p(b) = (q(p), g_p, R_p, q'(p))$, and the following hold:
  - start times are synchronized: $\mathsf{v}(t_{p_1}) = \mathsf{v}(t_{p_2})$, for each $p_1, p_2 \in \mathsf{dom}(b)$;
  - guards are satisfied: $\mathsf{v} \vDash g_p$, for each $p \in \mathsf{dom}(b)$;
  - resets are performed: $\mathsf{v}' = [\bigcup_{p \in \mathsf{dom}(b)} R_p]\mathsf{v}$;
  - invariants are satisfied: $\mathsf{v}' \models \mathsf{Inv}_p(q'(p))$, for each $p \in \mathsf{Proc}$;
  - other processes do not move: $q(p) = q'(p)$, for each $p \notin \mathsf{dom}(b)$.

A run in the local time semantics is a sequence of local steps:

$$(q_0, \mathsf{v}_0) \xrightarrow{\Delta_1} (q_0, \mathsf{v}_0') \xrightarrow{a_1} (q_1, \mathsf{v}_1) \xrightarrow{\Delta_2} (q_1, \mathsf{v}_1') \xrightarrow{a_2} \cdots \xrightarrow{a_n} (q_n, \mathsf{v}_n) \xrightarrow{\Delta_{n+1}} (q_n, \mathsf{v}_n')$$

where v_0 is an initial local valuation. Note that the valuations on a local run may not be synchronized. We make the standard assumption that $\mathsf{v}_0 \models \bigwedge_{p \in \mathsf{Proc}} \mathsf{Inv}_p(q_0(p))$.

A.2. Elimination of invariants

We now show that we can convert any network of timed automata to a network of timed automata without invariants, such that both the networks are equivalent w.r.t. reachability. We do this via a sequence of transformations, each of which eliminates some invariants from the network, while preserving the reachability properties of the network.

A.2.1 Initial and accepting states

In Section B, we show that we can transform any network of timed automata $\mathcal{N}$ into a *network in compact form* $\mathcal{N}'$ (see Appendix B), such that the reachability properties are preserved. Observe that the initial state and accepting states of each process of $\mathcal{N}'$ does not have state invariants. Therefore, without loss of generality, we can assume that the initial state and accepting states of each process of the given network does not have state invariants.

A.2.2 Restriction to upper-bound invariants

We will now show that we can eliminate invariants of the form $x \succ c$ from any network of timed automata. We define the size of a timed automaton as the sum of number of states and the total information associated to transitions. Concretely, the size of a timed automaton A is defined as the $|Q_A| + |T_A| \cdot g_{\max} \cdot X_A$, where Q_A, T_A, X_A are the set of states, transitions and clocks of A and $g_{\max}$ is the maximum number of atomic constraints in the guard of a transition in A.

Lemma A.1. *For every network of timed automata $\mathcal{N} = \{A_1, A_2 \cdots, A_n\}$, there exists a network of timed automata $\mathcal{N}' = \{A'_1, A'_2 \cdots, A'_n\}$, such that in each process A'_i of $\mathcal{N}'$, all invariants are of the form $x \prec c$, and both networks are equivalent w.r.t. reachability. The transformation takes time $\mathcal{O}(Q_{\max} \cdot T_{\max} \cdot X_{\max} \cdot n)$, where $Q_{\max} = \max_{1 \le i \le n} Q_i$, $T_{\max} = \max_{1 \le i \le n} T_i$ and $X_{\max} = \max_{1 \le i \le n} X_i$ where Q_i, T_i and X_i are the set of states, transitions and clocks of process A_i, respectively. The size of the network $\mathcal{N}'$ is greater than the size of $\mathcal{N}$ by $\mathcal{O}(T_{\mathcal{N}} \cdot |X|)$, where $T_{\mathcal{N}}$ is the set of transitions in $\mathcal{N}$ and $X = \bigcup_{1 \le i \le n} X_i$.*

Proof. Observe that lower-bound invariants (invariants of the form $x \succ c$) satisfy the property that once such an invariant is true, it remains true under time-elapse. So, it is sufficient to check the satisfaction of such invariants for the valuation with which the state is reached.

We propose a transformation to convert a timed automaton into a timed automaton without lower-bound invariants. Let A be a timed automaton

with state invariants. From the discussion in Section A.2.1, we can assume that the initial state of A does not have a state invariant. Pick any state q of A (other than the initial state). For each invariant $x \succ c$ of state q, we modify each transition of the form $p \xrightarrow[R]{a,g} q$ as follows:

- if $x \notin R$, then the guard g is replaced by the guard $g' \equiv g \wedge x \succ c$.
- otherwise,
  - if $c \leq 0$, then the guard is left unchanged, i.e., $g' \equiv g$.
  - if $c > 0$, then we remove the transition.

Finally, we remove the invariant $x \succ c$ associated to the state q. Repeat this transformation for each state of A, except the initial state.

Let $\mathcal{N}' = \{A'_1, \cdots, A'_n\}$ be the network obtained by carrying out the aforementioned transformation independently to each process of the network $\mathcal{N} = \{A_1, A_2, \cdots, A_n\}$. By definition, no process of $\mathcal{N}'$ does not have state invariants of the form $x \succ c$.

It is easy to see that for each valuation v, we have

$$\mathsf{v} \models g \text{ and } [R]\mathsf{v} \models x \succ c \;\Leftrightarrow\; \mathsf{v} \models g'$$

We can then prove that any local run of $\mathcal{N}$ is feasible in $\mathcal{N}'$, and vice versa. Both the directions of the proof follow by induction on the number of steps in the run, with the help of A.2.2. Specifically, we show that a run $(q_0, \mathsf{v}_0) \xrightarrow{\sigma} (q, \mathsf{v})$ of $\mathcal{N}$ is feasible in $\mathcal{N}'$, by inducting on the number of steps in the run. The converse direction also follows by a similar argument.

We now give a bound on the time taken for this transformation. Consider a state s of a process A_i of the network $\mathcal{N}$. We assume that s has only one invariant of the form $x \succ c$ for a given clock x; if there were two invariants of the form $x \succ c_1$ and $x \succ c_2$ in a state, where $c_1 > c_2$, we can ignore the invariant $x \succ c_2$. It is easy to see that, for the process A_i, the transformation takes $|Q_i|.|X_i|.|T_i|$ time. We carry out the transformation independently to each process of the network. Thus, the network $\mathcal{N}'$ can be constructed in time $\mathcal{O}(Q_{\max} \cdot T_{\max} \cdot X_{\max} \cdot n)$, where $Q_{\max} = \max\limits_{1 \leq i \leq n} Q_i$, $T_{\max} = \max\limits_{1 \leq i \leq n} T_i$ and $X_{\max} = \max\limits_{1 \leq i \leq n} X_i$.

Next, we give a bound on the size of the network $\mathcal{N}'$. The number of states and transitions in the new network $\mathcal{N}'$ is the same as that in $\mathcal{N}$. The only aspect in which $\mathcal{N}'$ is different from $\mathcal{N}$ is in the guards of the transitions and the invariants of states. Consider a transition in A_i with guard g. The new guard g' of this transition in A'_i can have X_i additional constraints. Consider a transition with guard g in $\mathcal{N}$ and guard g' in $\mathcal{N}'$. The guard g' can have up to X new atomic constraints more than the guard g. Thus, the size of the new network $\mathcal{N}'$ differs from the size of the original network $\mathcal{N}$

by $\mathcal{O}(T_{\mathcal{N}} \cdot |X|)$, where $T_{\mathcal{N}}$ is the number of transitions in $\mathcal{N}$ (and $\mathcal{N}'$) and $X = \bigcup_{1 \leq i \leq n} X_i$. □

Thus, without loss of generality, we can assume that the given network of timed automata is such that for each process of the network

- all invariants are of the form $x \prec c$.
- initial state and accepting states do not have state invariants.

Next, we present a transformation that eliminates upper-bound constraints from a given network of the aforementioned form, such that the new network is accepting iff the original network is accepting.

A.2.3 Elimination of upper-bound invariants

We consider a network $\mathcal{N} = \{A_1, A_2, \cdots, A_n\}$ of timed automata, where $A_i = (Q_i, \Sigma_i, X_i, T_i, q_i^0, F_i, \mathsf{Inv}_i)$. Recall that Inv_i associates an invariant from Φ_{X_i} to states from Q_i. We write $q(p)$ to denote the location of process A_p in the state q of $\mathcal{N}$. In other words, if the state q is a tuple of the form $(q_1, \cdots, q_n)$, then $q(p) = q_p$. We will abuse the notation to write $\mathsf{Inv}_p(q)$ to denote $\mathsf{Inv}_p(q(p))$.

We have already shown in Section A.2 that we can assume, without loss of generality, that each process of the network $\mathcal{N}$ has only invariants of the form $x \prec c$, and has no invariants in initial state or accepting states of processes. Thus, if $q \in F_i$, then we have $\mathsf{Inv}(q) \equiv \mathit{true}$.

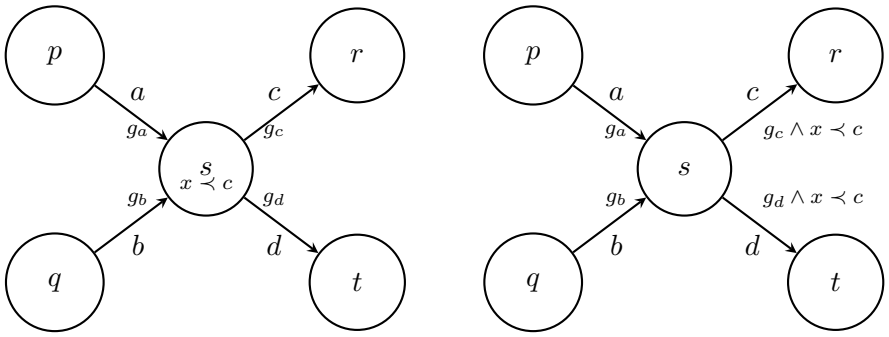
We will show that we can construct a new network $\mathcal{N}'$ by pushing all the invariants in each state of each process A_i of $\mathcal{N}$ to the guards on all the outgoing transitions. We now make the transformation precise. The required network is defined as $\mathcal{N}' = \{A'_1, \cdots, A'_n\}$ where the process A'_i is of the form $(Q_i, \Sigma_i, X_i, T'_i, q_i^0, F_i, \mathsf{Inv}'_i)$. The process A'_i of $\mathcal{N}'$ is obtained by transforming the process A_i of $\mathcal{N}$ as follows:

- For each state q of the process A_i of $\mathcal{N}$, we set $\mathsf{Inv}'_i(q) \equiv \mathit{true}$.
- For each transition $q \xrightarrow[R]{a,g} q'$ in process A_i, we have the transition $q \xrightarrow[R]{a,g'} q'$ in A'_i, where $g' \equiv g \wedge \mathsf{Inv}_i(q)$.

Thus, the process A'_i is different from A_i in only the guards of transitions and the invariants of states. The transformation is illustrated in Figure A.1.

We will now show that $\mathcal{N}'$ has an accepting run if and only if $\mathcal{N}$ has an accepting run.

Lemma A.2. *Each run in $\mathcal{N}$ is also feasible in $\mathcal{N}'$.*



(a) Process A of network $\mathcal{N}$ (b) Process A' of network $\mathcal{N}'$

Figure A.1: Transformation of a network to a network without state invariants

Proof. Consider a run σ of $\mathcal{N}$ of the form

$$(q_0, \mathsf{v}_0) \xrightarrow{\Delta_0} (q_0, \mathsf{v}'_0) \xrightarrow{a_1} (q_1, \mathsf{v}_1) \xrightarrow{\Delta_1} (q_1, \mathsf{v}'_1) \xrightarrow{a_2} \cdots \xrightarrow{\Delta_{l-1}} (q_{l-1}, \mathsf{v}'_{l-1}) \xrightarrow{a_l} (q_l, \mathsf{v}_l) \xrightarrow{\Delta_l} (q_l, \mathsf{v}'_l)$$

where v_0 is an initial local valuation and v'_l is a synchronized valuation.

We will show that σ is a run in $\mathcal{N}'$ as well.

Pick a configuration (q_i, v'_i), $0 \leq i \leq l$, in the run σ. We will show that $(q_i, \mathsf{v}'_i) \xrightarrow{a_{i+1}} (q_{i+1}, \mathsf{v}_{i+1})$ is feasible in $\mathcal{N}'$.

Suppose that the transition $q_i \xrightarrow{a_{i+1}} q_{i+1}$ has the guard g_p in process A_p, where $p \in \mathsf{dom}(a_{i+1})$. Similarly, suppose that the transition $q_i \xrightarrow{a_{i+1}} q_{i+1}$ in the new network $\mathcal{N}'$ has the guard g'_p in process A_p. From the definition of $\mathcal{N}'$, we know that $g'_p \equiv g_p \wedge \mathsf{Inv}_p(q_i)$.

From the run σ of $\mathcal{N}$, we know that $\mathsf{v}'_i \models \mathsf{Inv}_p(q_i)$ for all $p \in \mathsf{Proc}$. Further, since a_{i+1} is feasible from (q_i, v'_i), we know that for each $p \in \mathsf{dom}(a_{i+1})$, $\mathsf{v}'_i \models g_p$. From these observations, we can conclude that for each $p \in \mathsf{dom}(a_{i+1})$, we have $\mathsf{v}'_i \models g_p \wedge \mathsf{Inv}_p(q_i)$. This implies that a_{i+1} is feasible from (q_i, v'_i) in $\mathcal{N}'$ as well.

Further, observe that since $\mathcal{N}'$ has no state invariants, the delay Δ_i is feasible. So, the delay transition $(q_i, \mathsf{v}_i) \xrightarrow{\Delta} (q_i, \mathsf{v}'_i)$ is also feasible in $\mathcal{N}'$.

Thus, we have shown that each step of the run σ is feasible in $\mathcal{N}'$. □

Lemma A.3. *Each accepting run of $\mathcal{N}'$ is feasible in $\mathcal{N}$ as well.*

Proof. Consider an accepting run ρ of $\mathcal{N}'$.

$$(q_0, \mathsf{v}_0) \xrightarrow{\Delta_0} (q_0, \mathsf{v}'_0) \xrightarrow{a_1} (q_1, \mathsf{v}_1) \xrightarrow{\Delta_1} (q_1, \mathsf{v}'_1) \xrightarrow{a_2} \cdots \xrightarrow{\Delta_{l-1}} (q_{l-1}, \mathsf{v}'_{l-1}) \xrightarrow{a_l} (q_l, \mathsf{v}_l) \xrightarrow{\Delta_l} (q_l, \mathsf{v}'_l)$$

where v_0 is an initial local valuation and q_l is an accepting state and v'_l is a synchronized valuation.

We will show that the run ρ is feasible in $\mathcal{N}$.

From the definition of $\mathcal{N}'$, we know that the set of constraints present in each guard of $\mathcal{N}$ is strictly contained in the set of constraints present in the respective guard in $\mathcal{N}'$. As a consequence, if a_{i+1} is feasible from v'_i in $\mathcal{N}'$, then a_{i+1} is feasible from v'_i in $\mathcal{N}$ also. So, all that is left to show is that each valuation in the run satisfies the respective state invariant.

First, we show that it is enough to show that $\mathsf{v}'_i \models \mathsf{Inv}_p(q_i)$ for all $p \in \mathsf{Proc}$, where $0 \leq i \leq l$. This is a direct consequence of the fact that all the invariants in $\mathcal{N}$ are of the form $x \prec c$. Since $\mathsf{v}_i \xrightarrow{\Delta_i} \mathsf{v}'_i$, it is evident that $\mathsf{v}'_i \models \mathsf{Inv}_p(q_i)$ implies $\mathsf{v}_i \models \mathsf{Inv}_p(q_i)$.

We will now show that $\mathsf{v}'_i \models \mathsf{Inv}_p(q_i)$ for all $p \in \mathsf{Proc}$, where $0 \leq i \leq l$. Suppose that the transition $q_i \xrightarrow{a_{i+1}} q_{i+1}$ has the guard g_p and g'_p, in process A_p and A'_p, respectively. Recall that from the definition of $\mathcal{N}'$, we have $g'_p \equiv g_p \wedge \mathsf{Inv}_p(q_i)$.

Consider v'_i such that $0 \leq i \leq l$. We need to show that $\mathsf{v}'_i \models \mathsf{Inv}_p(q_i)$ for all $p \in \mathsf{Proc}$. Since a_{i+1} is feasible from v'_i, we know that for each $p \in \mathsf{dom}(a_{i+1})$, $\mathsf{v}'_i \models g'_p$. As a consequence, we have $\mathsf{v}'_i \models \mathsf{Inv}_p(q_i)$, for all $p \in \mathsf{dom}(a_{i+1})$.

It remains to show that $\mathsf{v}'_i \models \mathsf{Inv}_p(q_i)$ for $p \notin \mathsf{dom}(a_{i+1})$.

We have two cases for such a process A_p.

- Suppose that A_p participates in an action in ρ after a_{i+1}. Let a_{k+1} be the first such action, i.e., $p \in \mathsf{dom}(a_{k+1})$ for some $i < k < l$ such that $p \notin \mathsf{dom}(a_j)$ for $i < j \leq k$. Let the guard of a_{k+1} in A_p be g and in A'_p be g'. From the definition of $\mathcal{N}'$, we know that $g' \equiv g \wedge \mathsf{Inv}_p(q_k)$.

  Since a_{k+1} is feasible from v'_k, we know that $\mathsf{v}'_k \models g'$. As a consequence, we have $\mathsf{v}'_k \models \mathsf{Inv}_p(q_k)$. Further, we know that the process A_p has not participated in an action from (q_i, v'_i) to (q_k, v'_k). Consequently, the location of A_p is the same in q_i and q_k, i.e., $q_i(p) = q_k(p)$. Hence, we also have $\mathsf{Inv}_p(q_k) \equiv \mathsf{Inv}_p(q_i)$. Further, we know that $\mathsf{v}'_k(t_p) \geq \mathsf{v}'_i(t_p)$ and $\mathsf{v}'_k(\widetilde{x}) = \mathsf{v}'_i(\widetilde{x})$ for all $x \in X_p$. Since $\mathsf{v}'_k \models \mathsf{Inv}_p(q_k)$, we have $\mathsf{v}'_i \models \mathsf{Inv}_p(q_i)$.

- Suppose that A_p does not participate in any of the actions $a_{i+1}, \cdots, a_l$. But since the run ρ is accepting, this implies that the location of A_p in q_i is part of an accepting state. From our assumption, this implies that $\mathsf{Inv}_p(q_i) \equiv \mathit{true}$. As a consequence, in this case also, we have $\mathsf{v}'_i \models \mathsf{Inv}_p(q_i)$. □

From Lemma A.2 and Lemma A.3, we get Theorem A.1.

Theorem A.1. *$\mathcal{N}$ is accepting iff $\mathcal{N}'$ is accepting.*

The network $\mathcal{N}'$ can be obtained from $\mathcal{N}$ in time $\mathcal{O}(Q_{\max} \cdot T_{\max} \cdot X_{\max} \cdot n)$, where $Q_{\max} = \max_{1 \leq i \leq n} Q_i$, $T_{\max} = \max_{1 \leq i \leq n} T_i$ and $X_{\max} = \max_{1 \leq i \leq n} X_i$ where Q_i, T_i

and X_i are the set of states, transitions and clocks of process A_i, respectively. Further, the size of the network $\mathcal{N}'$ is greater than the size of $\mathcal{N}$ by $\mathcal{O}(T_{\mathcal{N}} \cdot |X|)$, where $T_{\mathcal{N}}$ is the set of transitions in $\mathcal{N}$ and $X = \bigcup_{1 \leq i \leq n} X_i$. The arguments for these results follow similarly to the proof of Lemma A.1.

Appendix B

Networks of timed automata in compact form

In this section, we show that any network of timed automata can be transformed into a network of timed automata that accepts by executing a special action, and has no invariants in initial and accepting states, such that reachability properties are preserved. Note that we work with networks of timed automata with state invariants in this section.

Definition B.1 (Network of timed automata in compact form)**.** A network of timed automata $\mathcal{N} = \{A_1, \cdots, A_n\}$ is said to be in *compact form* if it satisfies the following properties:

- $\mathcal{N}$ accepts by executing a special action α. In other words, if σ is an accepting run of $\mathcal{N}$, then the final action of σ is the special action α.
- For each process A_i of $\mathcal{N}$, neither the initial state nor the accepting states have state invariants.

We say that a network is *accepting* if it has an accepting run. We will show that we can convert any network $\mathcal{N}$ to a network $\mathcal{N}'$ in compact form, such that $\mathcal{N}$ is accepting iff $\mathcal{N}'$ is accepting. We will provide two different constructions, one for global-local systems and one for client-server systems. Note that the conversion technique for global-local systems is quite general, and can be applied to any network of timed automata. However, in the case of client-server systems, the resultant network in compact form is not a client-server system. Since we would like to apply our client-server POR technique on the network in the compact form, we propose here an alternate transformation that yields a client-server system in compact form.

B.1. Global-local systems

In this section, we will show that we can convert any global-local system $\mathcal{N}$ to a global-local system $\mathcal{N}'$ in compact form, such that $\mathcal{N}$ is accepting iff $\mathcal{N}'$ is accepting.

Let $\mathcal{N} = \{A_1, A_2, \cdots, A_n\}$ be a global-local system, where each process $A_i = (Q_i, \Sigma_i, X_i, T_i, q_i^0, F_i, \mathsf{Inv}_i)$ is a timed automaton with state invariants. For each process A_i of $\mathcal{N}$, let $A_i' = (Q_i', \Sigma_i', X_i', T_i', q_i'^0, F_i', \mathsf{Inv}_i')$ be the process obtained by adding

- new states Init_i and Final_i with no state invariants. In the process A_i', we have $Q_i' = Q_i \cup \{\mathsf{Init}_i, \mathsf{Final}_i\}$, $q_i'^0 = \mathsf{Init}_i$ and $F_i' = \{\mathsf{Final}_i\}$, $\mathsf{Inv}_i'(\mathsf{Init}_i) \equiv \mathit{true}$ and $\mathsf{Inv}_i'(\mathsf{Final}_i) \equiv \mathit{true}$.
- new global actions start and end, i.e., $\Sigma_i' = \Sigma_i \cup \{\mathsf{start}, \mathsf{end}\}$.
- the following transitions to T_i:
  - $\mathsf{Init}_i \xrightarrow[x \leq 0]{\mathsf{start}} q_i^0$, where x is some clock in X_i.
  - $q \xrightarrow{\mathsf{end}} \mathsf{Final}_i$, for all states $q \in F_i$.

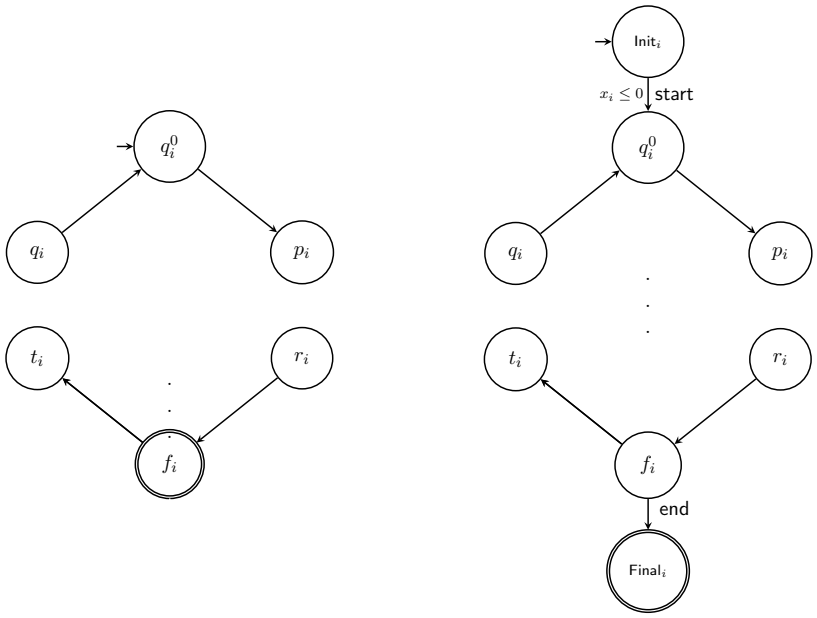
The transformation is illustrated in Figure B.1. Let $\mathcal{N}'$ be the network $\{A_1', A_2', \cdots, A_n'\}$. In a process A_i' of $\mathcal{N}'$, observe that the initial state and accepting states do not have invariants. Further, note that the only incoming transition to the accepting state of a process A_i' of $\mathcal{N}'$ is labelled by the action end. As a consequence, any accepting run of $\mathcal{N}'$ has to end by the execution of the global action end. Hence, the network $\mathcal{N}'$ is a global-local system in compact form.

Further, the guards on the corresponding transitions ensure that the global action start can only be executed from the initial valuation. Furthermore, the action end can only be executed from an accepting state of $\mathcal{N}$ and a synchronized valuation. As a consequence, we have Lemma B.1.

Lemma B.1. *$\mathcal{N}$ is accepting iff $\mathcal{N}'$ is accepting.*

Proof. Suppose that $\mathcal{N}$ is accepting. This means that there is a run σ of $\mathcal{N}$ of the form $(q_0, \mathsf{v}_0) \xrightarrow{\sigma} (q_l, \mathsf{v}_l)$, where q_l is an accepting state of $\mathcal{N}$, v_0 is an initial local valuation and v_l is a synchronized valuation. From the definition of $\mathcal{N}'$, we can conclude that the following run is feasible in $\mathcal{N}'$. Note that we write Init to denote the state $(\mathsf{Init}_1, \cdots, \mathsf{Init}_n)$ and Final to denote the state $(\mathsf{Final}_1, \cdots, \mathsf{Final}_n)$.

$$(\mathsf{Init}, \mathsf{v}_0) \xrightarrow{\mathsf{start}} (q_0, \mathsf{v}_0) \xrightarrow{\sigma} (q_l, \mathsf{v}_l) \xrightarrow{\mathsf{end}} (\mathsf{Final}, \mathsf{v}_l')$$



(a) Process A_i of network $\mathcal{N}$ (b) Process A_i' of network $\mathcal{N}'$

Figure B.1: Transformation of global-local system into a global-local system in compact form

Conversely, suppose that $\mathcal{N}'$ is accepting. We know that any accepting run of $\mathcal{N}'$ is of the form

$$(\mathsf{Init}, \mathsf{v}_0) \xrightarrow{\mathsf{start}} (q_0, \mathsf{v}_0') \xrightarrow{\rho} (q_l, \mathsf{v}_l) \xrightarrow{\mathsf{end}} (\mathsf{Final}, \mathsf{v}_l')$$

From the guards on the action start, we know that $\mathsf{v}_0' = \mathsf{v}_0$. Thus, we have the run $(q_0, \mathsf{v}_0) \xrightarrow{\rho} (q_l, \mathsf{v}_l)$ in $\mathcal{N}$, where q_0 is the initial state of $\mathcal{N}$ and v_0 is an initial local valuation. Further, from the definition of $\mathcal{N}'$, we know that the action end can only be executed from an accepting state of $\mathcal{N}$. Hence, q_l is an accepting state. Since end is a global action, it follows that v_l is a synchronized valuation. □

Note that if we start with a global-local system $\mathcal{N}$, then the network $\mathcal{N}'$ in compact form obtained as a result of the transformation is also a global-local system.

B.2. Client-server systems

In this section, we will show that we can convert any client-server system $\mathcal{N}$ to a client-server system $\mathcal{N}'$ in compact form, such that $\mathcal{N}$ is accepting iff $\mathcal{N}'$ is accepting. Note that the technique described in Section B.1 can

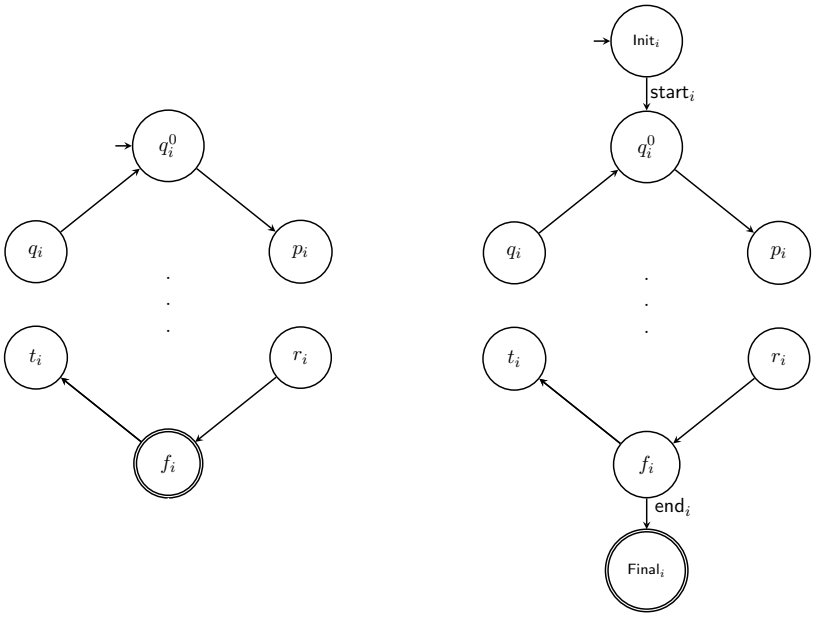
also be applied to client-server systems; however, in this case, the resultant network in compact form is not a client-server system. We would like to apply our client-server POR technique on the network in the compact form. In order to allow this, we propose here a different transformation that yields a client-server system in compact form.

Let $\mathcal{N} = \{S, C_1, C_2, \cdots, C_k\}$ be a client-server system, where the server process is given by $S = (Q_s, \Sigma_s, X_s, T_s, q_s^0, F_s, \mathsf{Inv}_s)$ and each client process is of the form $C_i = (Q_i, \Sigma_i, X_i, T_i, q_i^0, F_i, \mathsf{Inv}_i)$.

We first give the transformation of the client processes. For each client C_i of $\mathcal{N}$, let $C_i' = (Q_i', \Sigma_i', X_i', T_i', q_i'^0, F_i', \mathsf{Inv}_i')$ be the process obtained by adding

- new states Init_i and Final_i with no state invariants. Thus, In C_i', we have $Q_i' = Q_i \cup \{\mathsf{Init}_i, \mathsf{Final}_i\}$, $q_i'^0 = \mathsf{Init}_i$ and $F_i' = \{\mathsf{Final}_i\}$, $\mathsf{Inv}_i'(\mathsf{Init}_i) \equiv \mathit{true}$ and $\mathsf{Inv}_i'(\mathsf{Final}_i) \equiv \mathit{true}$.
- new communication actions start_i and end_i; $\Sigma_i' = \Sigma_i \cup \{\mathsf{start}_i, \mathsf{end}_i\}$.
- the following transitions to T_i:
  - $\mathsf{Init}_i \xrightarrow{\mathsf{start}_i} q_i^0$.
  - $q \xrightarrow{\mathsf{end}_i} \mathsf{Final}_i$, for all states $q \in F_i$.

The transformation is illustrated in Figure B.2. Next, we give the transfor-

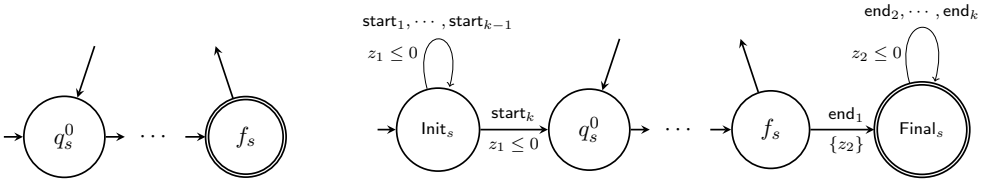


(a) Client C_i of network $\mathcal{N}$ (b) Client C_i' of network $\mathcal{N}'$

Figure B.2: Transformation of client-server system into a client-server system in compact form: client processes

mation of the server process. Let $S' = (Q'_s, \Sigma'_s, X'_s, T'_s, q'^0_s, F'_s, \mathsf{Inv}'_s)$ be the process obtained by adding to the server process S of $\mathcal{N}$

- new states Init_s and Final_s with no state invariants. Thus, we have $Q'_s = Q_s \cup \{\mathsf{Init}_s, \mathsf{Final}_s\}$, $q'^0_s = \mathsf{Init}_s$ and $F'_s = \{\mathsf{Final}_s\}$, $\mathsf{Inv}'_s(\mathsf{Init}_s) \equiv \mathit{true}$ and $\mathsf{Inv}'_s(\mathsf{Final}_s) \equiv \mathit{true}$.
- new communication actions $\mathsf{start}_1, \mathsf{start}_2, \cdots, \mathsf{start}_k$ and end_1, end_2, $\cdots$, end_k, i.e., $\Sigma'_s = \Sigma_s \cup \bigcup_{1 \le j \le k} \{\mathsf{start}_j, \mathsf{end}_j\}$.
- new clocks z_1, z_2. Thus, $X'_s = X_s \cup \{z_1, z_2\}$.
- the following transitions to T_s:
  - $\mathsf{Init}_s \xrightarrow[z_1 \le 0]{\mathsf{start}_j} \mathsf{Init}_s$, for $1 \le j < k$.
  - $\mathsf{Init}_s \xrightarrow[z_1 \le 0]{\mathsf{start}_k} q^0_s$.
  - $q \xrightarrow[\{z_2\}]{\mathsf{end}_1} \mathsf{Final}_s$, for all states $q \in F$.
  - $\mathsf{Final}_s \xrightarrow[z_2 \le 0]{\mathsf{end}_j} \mathsf{Final}_s$, for $1 < j \le k$.



(a) Server S of network $\mathcal{N}$ (b) Server S' of network $\mathcal{N}'$

Figure B.3: Transformation of client-server system into a client-server system in compact form: server process

The transformation described above is illustrated in Figure B.3.

Let $\mathcal{N}'$ be the network $\{S', C'_1, C'_2, \cdots, C'_k\}$. It is clear that the initial state and accepting states of processes in $\mathcal{N}'$ do not have state invariants. Further, from definition of $\mathcal{N}'$, the only incoming transition to the accepting state F'_i of A'_i is labelled by the action end_i. Moreover, from the definition of guards of the actions $\mathsf{end}_1, \cdots, \mathsf{end}_k$ in the server process S', it follows that these actions can only be executed consecutively, in 0 time. Hence, any accepting path of $\mathcal{N}'$ must end with the sequence of communication actions $\mathsf{end}_1 \cdots \mathsf{end}_k$. Therefore, $\mathcal{N}'$ is a client-server system in compact form, with the sequence of actions end_k playing the role of the special action α.

Next, observe that the guards on the corresponding transitions ensure that the start actions can only be executed in 0 time. Similarly, the end

actions can only be executed in 0 time from some (q, v), where q is an accepting state. Since the end actions are communication actions that can only be executed in 0 time, it is guaranteed that v is a synchronized valuation. As a consequence, we have Lemma B.2.

Lemma B.2. *$\mathcal{N}$ is accepting iff $\mathcal{N}'$ is accepting.*

Proof. Suppose that $\mathcal{N}$ is accepting. This means that there is a run σ of $\mathcal{N}$ of the form $(q_0, \mathsf{v}_0) \xrightarrow{\sigma} (q_l, \mathsf{v}_l)$, where q_l is an accepting state of $\mathcal{N}$, v_0 is an initial local valuation and v_l is a synchronized valuation. From the definition of $\mathcal{N}'$, we know that the following run is feasible in $\mathcal{N}'$. Note that we write Init to denote the state $(\mathsf{Init}_s, \mathsf{Init}_1, \cdots, \mathsf{Init}_k)$ and Final to denote the state $(\mathsf{Final}_s, \mathsf{Final}_1, \cdots, \mathsf{Final}_k)$.

$$(\mathsf{Init}, \mathsf{v}_0) \xrightarrow{\mathsf{start}_1} \xrightarrow{\mathsf{start}_2} \cdots \xrightarrow{\mathsf{start}_k} (q_0, \mathsf{v}_0) \xrightarrow{\sigma} (q_l, \mathsf{v}_l) \xrightarrow{\mathsf{end}_1} \xrightarrow{\mathsf{end}_2} \cdots \xrightarrow{\mathsf{end}_k} (\mathsf{Final}, \mathsf{v}'_l)$$

Conversely, suppose that $\mathcal{N}'$ is accepting. We know that any accepting run of $\mathcal{N}'$ is of the form

$$(\mathsf{Init}, \mathsf{v}_0) \xrightarrow{\mathsf{start}_1} \xrightarrow{\mathsf{start}_2} \cdots \xrightarrow{\mathsf{start}_k} (q_0, \mathsf{v}'_0) \xrightarrow{\rho} (q_l, \mathsf{v}_l) \xrightarrow{\mathsf{end}_1} \xrightarrow{\mathsf{end}_2} \cdots \xrightarrow{\mathsf{end}_k} (\mathsf{Final}, \mathsf{v}'_l)$$

From the guards on the actions $\mathsf{start}_1, \cdots, \mathsf{start}_k$, we know that no time is elapsed and no clocks are reset between v_0 and v'_0. We have the run $(q_0, \mathsf{v}'_0) \xrightarrow{\rho} (q_l, \mathsf{v}_l)$ in $\mathcal{N}$, where q_0 is the initial state and v'_0 is an initial local valuation. From the definition of $\mathcal{N}'$, we know that

- the action end_i is only enabled from an accepting state of A_i.
- the actions $\mathsf{end}_1, \cdots, \mathsf{end}_k$ are executed in 0 time.

As a consequence, we can conclude that q_l is an accepting state. Further, since the actions $\mathsf{end}_1, \cdots, \mathsf{end}_k$ are communication actions that are executed in 0 time, it is guaranteed that v_l is a synchronized valuation. Therefore, the run ρ is an accepting run of $\mathcal{N}$. □

We remark that if we start with a client-server system $\mathcal{N}$, then the network $\mathcal{N}'$ in compact form obtained as a result of the transformation is also a client-server system.

Appendix C

Sync-subsumption check for local zones

In this section, we give a direct test to check *sync-subsumption* (see Definition 5.5) between local zones.

$\sqsubseteq^{\mathfrak{a}LU}_{\mathsf{sync}}$ inclusion test for local zones

We say that a local zone Z is sync-subsumed by local zone Z', denoted $\mathsf{Z} \sqsubseteq^{\mathfrak{a}LU}_{\mathsf{sync}} \mathsf{Z}'$, if

$$\mathsf{global}(\mathsf{sync}(\mathsf{Z})) \sqsubseteq^{\mathfrak{a}}_{LU} \mathsf{global}(\mathsf{sync}(\mathsf{Z}'))$$

The naive test to check sync-subsumption involves the following steps:

- Computing $\mathsf{sync}(\mathsf{Z})$ and $\mathsf{sync}(\mathsf{Z}')$. From Lemma 4.9, we know that upon synchronizing a local zone, we still have a local zone.
- Converting $\mathsf{sync}(\mathsf{Z})$ and $\mathsf{sync}(\mathsf{Z}')$ to offset zones by applying the global operation. By Lemma 4.9, the operator global converts a local zone into a global zone. Let $\mathcal{Z} = \mathsf{global}(\mathsf{sync}(\mathsf{Z}))$ and $\mathcal{Z}' = \mathsf{global}(\mathsf{sync}(\mathsf{Z}'))$. We further apply the std operator on the offset zones $\mathcal{Z}$ and $\mathcal{Z}'$ to get standard zones Z and Z' respectively.
- Applying the $\sqsubseteq^{\mathfrak{a}}_{LU}$ check for Z and Z'.

We will now propose a simpler and more direct test to detect this.

We denote by Z_{xy} the weight of the edge $x \to y$ in the distance graph of Z. We now recall the standard test to check if $Z \sqsubseteq^{\mathfrak{a}}_{LU} Z'$.

Lemma C.1. *[HKSW11] Let Z and Z' be non-empty standard zones. Then, $Z \not\sqsubseteq^{\mathfrak{a}}_{LU} Z'$ if and only if there exist two clock variables x and y, such that*

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

We now propose an equivalent condition for global zones in the offset setting using the translation of offset zones to standard zones. Recall the translation of offset zones to standard zones as given in Definition 2.33. From this translation and Lemma C.1, we get the following $\sqsubseteq^{\mathfrak{a}}_{LU}$ subsumption check for offset zones:

Lemma C.2. *Let $\mathcal{Z}$ and $\mathcal{Z}'$ be non-empty offset zones. Then, $\mathcal{Z} \not\sqsubseteq^{\mathfrak{a}}_{LU} \mathcal{Z}'$ if and only if there exist two offset clock variables $\widetilde{x}$ and $\widetilde{y}$, such that*

$$\mathcal{Z}_{t\widetilde{x}} \geq (\leq, -U_x) \text{ and } \mathcal{Z}'_{\widetilde{y}\widetilde{x}} < \mathcal{Z}_{\widetilde{y}\widetilde{x}} \text{ and } \mathcal{Z}'_{\widetilde{y}\widetilde{x}} + (<, -L_y) < \mathcal{Z}_{t\widetilde{x}}$$

Henceforth, we work with this condition over offset zones, rather than the former condition over standard zones.

Next, we show that synchronized zones can be efficiently computed from local time-elapsed zones, as given by the following lemma.

Lemma C.3. *Given a local-time elapsed zone Z, the distance graph of $\mathsf{sync}(\mathsf{Z})$ in the canonical form, is obtained from the distance graph of Z by the following steps:*

- *Set the weight of all $t_i \to t_j$ edges to $(\leq, 0)$*
- *Set the weight of all $t_i \to \widetilde{x}$ edge to $\min\{G^{\mathsf{Z}}_{t_i,\widetilde{x}} \mid t_i \in T\}$*

Proof. Let Z be a local-time elapsed zone and let G_{Z} be the canonical distance graph of Z. Since Z is local-time elapsed, we know that all edges of the form $\widetilde{x} \to t$ for $\widetilde{x} \in \widetilde{X}$ and $t \in T$ has weight $(<, \infty)$.

Let $\mathsf{Z}' = \mathsf{Z} \wedge \bigwedge_{t_i,t_j \in T}(t_i = t_j)$. Each edge in $G_{\mathsf{Z}'}$ of the form $t_i \to t_j$ for $t_i, t_j \in T$ has weight (≤ 0). Let Z'' be the zone obtained by canonicalizing Z'. We consider the edges of $G_{\mathsf{Z}''}$. Recall that we say a path is *lighter* than another, if the weight of the former is lesser than the weight of the latter path.

$\widetilde{x} \to t_i$ The lightest path of this form, where $\widetilde{x} \in \widetilde{X}$ to any $t_i \in T$ is $(<, \infty)$. Consider a lighter path from $\widetilde{x}$ to t. It must be of the form $\widetilde{x} \to \cdots \to \widetilde{y} \to t_j \to \cdots \to t_i$. Since the edge $\widetilde{y} \to t_j$ has weight $(<, \infty)$, this path has weight $(<, \infty)$.

$t_i \to t_j$ The lightest path between any two reference clocks t_i and t_j is $(\leq, 0)$. We know that the weight of the edge $t_i \to t_j$ in $G_{\mathsf{Z}'}$ is $(\leq, 0)$. Now, suppose that there is a lighter path in $G_{\mathsf{Z}''}$. Since all edges between reference clocks have weight $(\leq, 0)$, this path can only be of the form $t_i \to \cdots \to \widetilde{x} \to t_j$. Once again, since each edge $\widetilde{x} \to t_j$ has weight $(<, \infty)$, this path also has weight $(<, \infty)$.

$\widetilde{x} \to \widetilde{y}$ The lightest path between two offset clocks is given by the direct edge between them from $G_{\mathsf{Z}'}$.

$t_i \to \widetilde{x}$ We will show that the weight of such an edge is equal to $\min\{Z_{t_j,\widetilde{x}} \mid t_j \in T\}$. Consider paths of the form $t_i \xrightarrow{(\leq,0)} t_j \xrightarrow{\preceq,c} \widetilde{x}$. The minimum weight of such a path is given by $\min\{Z_{t_j,\widetilde{x}} \mid t_j \in T\}$.

Suppose that there is a lighter path from t_i to $\widetilde{x}$. Note that this path cannot use edges of the form $\widetilde{y} \to t_j$, as these edges have weight $(<,\infty)$.

Suppose that this path is of the from $t_i \xrightarrow{(\leq,0)} \cdots \xrightarrow{(\leq,0)} t_j \to \cdots \to \widetilde{x}$. where the $t_i \xrightarrow{(\leq,0)} \cdots \xrightarrow{(\leq,0)} t_j$ part of the path consists entirely of reference clocks. The part involving reference clocks can be substituted by the simple edge $t_i \xrightarrow{(\leq,0)} t_j$. Further, observe that all edges from t_j to $\widetilde{x}$ are from G_{Z}. Since G_{Z} is canonical, this part of the path can be substituted by the simple edge $t_j \to \widetilde{x}$. Then, the path reduces to $t_i \xrightarrow{(\leq,0)} t_j \to \widetilde{x}$. It is easy to see that this path cannot be lighter. □

This is a huge improvement over the naive computation of synchronized zones, which would first involve adding edges of weight 0 between reference clocks and then canonicalizing it. Let n be the number of processes and c be the number of clocks in the network. The naive computation takes time $\mathcal{O}(c+n)^3$, whereas the computation given in Lemma C.3 takes time $\mathcal{O}(n^2 + c \cdot n^2) = \mathcal{O}(c \cdot n^2)$.

We will now use the results given in Lemma C.3 and Lemma C.2 to propose a direct check for sync-subsumption between local zones, without transforming the local zones. Since the computation of $\mathsf{sync}(\mathsf{Z})$ from a local zone Z requires only specific local transformations of some clocks (as stated in Lemma C.3), we can execute the check specified by Lemma C.2 directly using the edges of the distance graphs of the local zone Z and Z', as stated in the following lemma.

Lemma C.4. *Let* Z *and* Z' *be non-empty time-elapsed local zones. Then,* Z *is not sync-subsumed by* Z' *if and only if there exist two variables* $\widetilde{x}$ *and* $\widetilde{y}$ *such that*

- $\min\{\mathsf{Z}_{t_i,\widetilde{x}} \mid t_i \in T\} \geq (\leq, -U_x)$ *and*
- $\mathsf{Z}'_{\widetilde{y}\widetilde{x}} < \mathsf{Z}_{\widetilde{y}\widetilde{x}}$ *and*
- $\mathsf{Z}'_{\widetilde{y}\widetilde{x}} + (<, -L_y) < \min\{\mathsf{Z}_{t_i,\widetilde{x}} \mid t_i \in T\}$

Proof. Let Z and Z' be non-empty time-elapsed local zones. Let $\mathcal{Z} = \mathsf{global}(\mathsf{sync}(\mathsf{Z}))$ and $\mathcal{Z}' = \mathsf{global}(\mathsf{sync}(\mathsf{Z}'))$. We denote the distance graphs of $\mathcal{Z}, \mathcal{Z}', \mathsf{Z}$ and Z', by $G_{\mathcal{Z}}, G_{\mathcal{Z}'}, G_{\mathsf{Z}}$ and $G_{\mathsf{Z}'}$, respectively.

From Lemma C.2, we know that $\mathcal{Z} \not\sqsubseteq^{\mathfrak{a}}_{LU} \mathcal{Z}'$, if there do not exist two offset clock variables $\widetilde{x}, \widetilde{y}$ such that

$$\mathcal{Z}_{t\widetilde{x}} > (\leq, -U_x) \text{ and } \mathcal{Z}'_{\widetilde{y}\widetilde{x}} < \mathcal{Z}_{\widetilde{y}\widetilde{x}} \text{ and } \mathcal{Z}'_{\widetilde{y}\widetilde{x}} + (<, -L_y) < \mathcal{Z}_{t\widetilde{x}}$$

The key observation is that weights of the relevant edges of $G_{\mathcal{Z}}$ and $G_{\mathcal{Z}'}$ can be directly obtained from G_{Z} and $G_{\mathsf{Z}'}$, without modifying G_{Z} and $G_{\mathsf{Z}'}$. From Lemma C.3, we know that the weight of the $t \to \widetilde{x}$ edge in $G_{\mathcal{Z}}$, where $\mathcal{Z} = \mathsf{global}(\mathsf{sync}(\mathsf{Z}))$, is given by $\min\{G^{\mathsf{Z}}_{t_i,\widetilde{x}} \mid t_i \in T\}$ in G_{Z}. Similarly, the weight of the $\widetilde{y} \to \widetilde{x}$ edge in $G_{\mathcal{Z}}$ is equal to the weight of the $\widetilde{y} \to \widetilde{x}$ edge in G_{Z}.

As a consequence, the weights of the relevant edges of $G_{\mathcal{Z}}$ can be obtained from G_{Z} - either directly, in the case of $\widetilde{x} \to \widetilde{y}$ edges and or by finding the minimum of all $t_i \to \widetilde{x}$ edges for the $t \to \widetilde{x}$ edges.

Further, note that the only edges required from $G_{\mathcal{Z}'}$ are the $\widetilde{y} \to \widetilde{x}$ edges, which is the same as the respective $\widetilde{y} \to \widetilde{x}$ edge in $G_{\mathsf{Z}'}$. Therefore, these edges may be directly taken from $G_{\mathsf{Z}'}$. □

Gain over the naive approach

Let n denote the number of processes in the network and $c = |X|$ be the number of clocks in the network. Then, the check given in Lemma C.4 takes $\mathcal{O}(c^2 + n \cdot c)$ time and requires constant amount of space.

Running time. As already discussed, the naive test to check if a local zone Z is sync-subsumed by another local zone Z' has the following steps:

1. compute $\mathsf{sync}(\mathsf{Z})$ and $\mathsf{sync}(\mathsf{Z}')$ - the naive sync-zone computation takes $\mathcal{O}(c+n)^3$ time.
2. Convert $\mathsf{sync}(\mathsf{Z})$ and $\mathsf{sync}(\mathsf{Z}')$ to offset zones by the global operation and then to standard zones by the std operation - can be done by a $\mathcal{O}(c^2 + c \cdot n)$ procedure.
3. Applying the $\sqsubseteq^{\mathsf{a}}_{LU}$ check on the offset zones obtained in the previous step - this procedure takes $\mathcal{O}((c+1)^2)$ time.

As can be seen above, the naive $\sqsubseteq^{\mathsf{a}LU}_{\mathsf{sync}}$ subsumption check involves five steps - applying procedure (1) and (2) on both the zones Z and Z' and then doing the check (3) on the standard zones obtained. The time complexity of the procedure is $\mathcal{O}(c+n)^3$. On the other hand, the explicit check given in Lemma C.3, involves only one step which takes $\mathcal{O}(c^2 + c \cdot n)$ time.

Furthermore, while the naive check would involve computation over both Z and Z', in the direct check, no computation needs to be done on Z'. This is because the only edge weights from Z' required for the test are of edges of the form $x \to y$. Since the weight of these edges is the same as weights of the respective $\widetilde{y} \to \widetilde{x}$ edges in the distance graph of Z', these edge weights may be directly obtained from the distance graph of Z'.

Space complexity The naive $\sqsubseteq^{\mathfrak{a}LU}_{\mathsf{sync}}$ subsumption check also uses additional space of the order of $\mathcal{O}((c+n)^2)$ since we are required to store the distance graphs of $\mathsf{sync}(\mathsf{Z})$ and $\mathsf{sync}(\mathsf{Z}')$, and in the subsequent step, $\mathsf{std}(\mathsf{global}(\mathsf{sync}(\mathsf{Z})))$ and $\mathsf{std}(\mathsf{global}(\mathsf{sync}(\mathsf{Z}')))$. In contrast, the explicit check given in Lemma C.3 needs only constant space. The only part of the check that requires space involves computing the minimum of weights of all $t_i \to \widetilde{x}$ edges in the distance graph of Z, for $i \in \{1, 2, \cdots, k\}$ - this computation only requires constant space.

In practice, whenever a new zone Z is visited while exploring the local zone graph, we would need to compute $\mathsf{sync}(\mathsf{Z})$ and then the standard zone $Z = \mathsf{std}(\mathsf{global}(\mathsf{sync}(\mathsf{Z})))$ and store the distance graph of these zones, in addition to the distance graph of the local zone. The algorithm would need to store this standard zone Z for every local zone Z that has been seen, and whenever a new zone Z' is seen, check if this zone is $\sqsubseteq^{\mathfrak{a}LU}_{\mathsf{sync}}$ subsumed by some zone that has already been seen. In contrast, if the direct $\sqsubseteq^{\mathfrak{a}LU}_{\mathsf{sync}}$ check is used, there is no need to store any of this additional information for local zones - this is because any relevant edge weight that is required for subsumption checks can be directly obtained from the distance graphs of the local zones.