

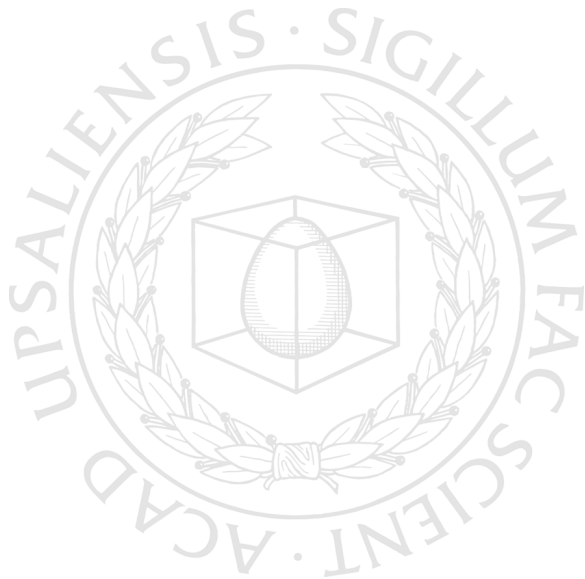


UPPSALA
UNIVERSITET

*Digital Comprehensive Summaries of Uppsala Dissertations
from the Faculty of Science and Technology 633*

Infinite Structures in Timed Systems

PAVEL KRCAL



ACTA
UNIVERSITATIS
UPSALIENSIS
UPPSALA
2009

ISSN 1651-6214
ISBN 978-91-554-7496-6
urn:nbn:se:uu:diva-100549

Dissertation presented at Uppsala University to be publicly examined in 2446, Lägerhyddsvägen 2, Uppsala, Friday, May 15, 2009 at 13:15 for the degree of Doctor of Philosophy. The examination will be conducted in English.

Abstract

Krčál, P. 2009. Infinite Structures in Timed Systems. Acta Universitatis Upsaliensis. *Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology* 633. 43 pp. Uppsala. ISBN 978-91-554-7496-6.

Real time systems distinguish themselves by explicitly stating timing constraints in the system specification. This requires specific methods and tools in system design to ensure such constraints. We focus on one of the methods applied in the validation phase, namely formal verification. This method automatically establishes correctness of the system model with mathematical rigor. In order to apply mechanical procedures to determine whether the system satisfies the requirements, we first have to model the validated part of the system in a mathematical form. This thesis deals with one such formalism - timed automata - and investigates different types of infinite state structures arising in the verification procedures related to this formalism. There are two different views which open the door for introduction of such structures.

First, we turn outwards and extend timed automata with additional infinite data structures - unbounded queues. These queues serve different purposes. In one case, the queues contain computation tasks and, together with a timed automaton, model a real-time system with tasks. The problem of interest in this setting is schedulability analysis. We investigate the decidability boundary in presence of various features such as preemption, variable computation times of tasks, and communication between the timed automaton and the task queue. In the other case, we use queues for asynchronous communication between timed automata running synchronously in parallel. These queues store messages issued by one automaton and waiting to be read by another automaton. Such situations occur among other cases in real-time control systems where several concurrently running tasks communicate via buffers. We study the decidability border for reachability analysis depending on various communication topologies of these systems.

Secondly, we turn inwards and study a peculiar feature of timed automata which allows them to enforce behaviors where time distances between events monotonically grow while being bounded by some integer. This feature can be characterized by unbounded counters recording the number of such enforced increases. When we switch from the dense time semantics used for modeling to an implementation with a fixed clock rate (sampled semantics), only behaviors which correspond to a bounded usage of these counters are preserved. We describe operation of these counters as a new type of a counter automaton and prove that one can effectively check whether the counters are used in a bounded way. As a result, it is possible to check for a given timed automaton whether there is an implementation with a fixed sampling rate which preserves all qualitative behaviors.

Keywords: timed automata, scheduling, queues, sampling, finite automata with counters, limitedness

Pavel Krčál, Department of Computer Systems, Box 337, Uppsala University, SE-75105 Uppsala, Sweden

© Pavel Krčál 2009

ISSN 1651-6214

ISBN 978-91-554-7496-6

urn:nbn:se:uu:diva-100549 (<http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-100549>)

To Anne

List of Papers

This thesis is based on the following papers, which are referred to in the text by their Roman numerals.

- I **Task Automata: Schedulability, Decidability and Undecidability.** Elena Fersman, Pavel Krcal, Paul Pettersson and Wang Yi. *Journal of Information and Computation*, 205(8):1149–1172, 2007.
- II **R-automata.** Parosh Aziz Abdulla, Pavel Krcal and Wang Yi. Technical Report 2008-016, Uppsala University 2008. A short version of this paper appeared in *CONCUR'08: Proceedings of the 19th International Conference on Concurrency Theory*, volume 5201 of *Lecture Notes in Computer Science*, pages 67–81, Springer-Verlag, 2008.
- III **Sampled Semantics of Timed Automata.** Parosh Aziz Abdulla, Pavel Krcal and Wang Yi. Submitted to *Journal of Logical Methods in Computer Science*. A preliminary version of this paper appeared in *FoS-SaCS'07: Proceedings of the 10th International Conference on Foundations of Software Science and Computation Structures*, volume 4423 of *Lecture Notes in Computer Science*, pages 2–16, Springer-Verlag, 2007.
- IV **Communicating Timed Automata: The More Synchronous, the More Difficult to Verify.** Pavel Krcal and Wang Yi. Technical Report 2006-006, Uppsala University 2006. A short version of this paper appeared in *CAV'06: Proceedings of the 18th International Conference on Computer Aided Verification*, volume 4144 of *Lecture Notes in Computer Science*, pages 243–257, Springer-Verlag, 2006.

Comments on my Participation:

- I I have shown the undecidability result, written most of the proofs and the undecidability part, and proposed the definition of scheduling policies.
- II I have suggested the model, participated in discussions, proved the results and written the paper.
- III I have formulated the problem, participated in discussions, proved the results and written the paper.
- IV I have participated in discussions, formulated the problems, proved the results and written the major parts of the paper.

Other Publications

- **Universality of R-automata with Value Copying.** Parosh Aziz Abdulla, Pavel Krcal and Wang Yi. In *INFINITY'08: Proceedings of the 10th International Workshop on Verification of Infinite-State Systems*, 2008, to appear.
- **Multi-Processor Schedulability Analysis of Preemptive Real-Time Tasks with Variable Execution Times.** Pavel Krcal, Martin Stigge and Wang Yi. In *FORMATS'07: Proceedings of the 5th International Conference on Formal Modelling and Analysis of Timed Systems*, volume 4763 of *Lecture Notes in Computer Science*, pages 274–289, Springer-Verlag, 2007.
- **Sampled Universality of Timed Automata.** Parosh Aziz Abdulla, Pavel Krcal and Wang Yi. In *FoSSaCS'07: Proceedings of the 10th International Conference on Foundations of Software Science and Computation Structures*, volume 4423 of *Lecture Notes in Computer Science*, pages 2–16, Springer-Verlag, 2007.
- **On Sampled Semantics of Timed Systems.** Pavel Krčál and Radek Pelánek. In *FSTTCS'05: Proceedings of the 25th International Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 3821 of *Lecture Notes in Computer Science*, pages 310–321, Springer-Verlag, 2005.
- **Timed vs. Time-Triggered Automata.** Pavel Krcal, Leonid Mokrushin, P.S. Thiagarajan, and Wang Yi. In *CONCUR'04: Proceedings of the 15th International Conference on Concurrency Theory*, volume 3170 of *Lecture Notes in Computer Science*, pages 340–354, Springer-Verlag, 2004.
- **Decidable and Undecidable Problems in Schedulability Analysis Using Timed Automata.** Pavel Krcal and Wang Yi. In *TACAS'04: Proceedings of the 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 2988 of *Lecture Notes in Computer Science*, pages 236–250, Springer-Verlag, 2004.

Acknowledgements

First and foremost, I would like to thank my supervisor Wang Yi for all his support, advises, and patience. I am especially grateful for the great deal of freedom which I enjoyed, combined with encouragement in difficult moments. Of equal importance for me was the atmosphere of trust which he maintained.

I would like to thank Parosh Abdulla for long discussions, patient listening to my proof ideas, and his research enthusiasm. I am also grateful to Paul Pettersson for various discussions and practical help.

I have spent one research month at National University of Singapore. I thank P.S. Thiagarajan for inviting me there, Shaofa Yang for taking care of the practical matters, and both of them for the exciting research I could take part in.

My thanks and gratitude also belong to my undergraduate supervisor Luboš Brim and other professors from the ParaDiSe laboratory at the Department of Informatics, Masaryk University Brno, Czechia, who sent me to Sweden perfectly prepared for graduate studies.

Two of my friends and colleagues, Radek Pelánek and Rafał Somla, substantially helped me with their direct, open, and precise feedback. I have learned a lot from Radek by our collaboration during his visit in Uppsala and numerous discussions in Brno. Rafał never rejected to listen to my ideas and to take the effort of spotting their weak points.

Many thanks go to past and present members of our lunch group: David Eklov, Olga Grinchtein, John Håkansson, Leonid Mokrushin, Guan Nan, Sven Sandberg, Martin Stigge, and Simon Tschirner.

I received a lot of support from my friends in Uppsala who showed significantly greater interest in, e.g., skating, cooking, hiking, and classical music, than abstract constructions of theoretical computer science. I really enjoyed all things which we did together. Even though I cannot name all, I would like to thank explicitly at least some: Vítěk Kříž, Michiel van Lun, Jiří Novák, Marian Novotný, Pavel Plevka, Josef Seibt, and the past and present members of the whist group.

I thank Justin Pearson, Jonathan Cederberg, Daniel Ying and Anne Wuttke for their help. Justin corrected my English in Part I, I am to be blamed for the remaining mistakes and clumsy expressions. Jonathan translated my summary to Swedish and Daniel helped me with updating it. Anne drew the clockwork on the thesis cover.

I owe special thanks to my parents and brothers. Knowing that you are there made everything easier. Last, but not least, I would like to thank to my fiancée Anne for all her love.

Sammanfattning på svenska

Datorer inklusive datorkraft som är integrerad i andra produkter kan ses som maskiner som beräknar svar på våra frågor. Beräkningarna är framgångsrika i de fall där ett korrekt svar fås inom rimlig tid. Detta innebär att frågan fortfarande är relevant när svaret returneras. För flertalet frågor kan deadline eller andra tidsmässiga krav summeras som *ju fortare desto bättre*, utan att ge några uttryckliga begränsningar. Det framgår av sammanhanget vad som är kompromissen mellan hastighet och andra relevanta mått och man har flertalet acceptabla valmöjligheter. Det finns även flertalet andra frågor utan utrymme för sådan flexibilitet. I dessa fall har vi konkreta begränsningar på svarstider och liknande som en explicit del av problemformuleringen, och vi har att göra med vad som kallas *realtidssystem*.

Det faktum att timing är en explicit del av problemformuleringen ger upphov till nya problem, angreppssätt och lösningar i designen av realtidssystem. Detta faktum kan försvåra arbetet genom ökad komplexitet, men också göra att man kan använda sig av speciella metoder och verktyg. För detta ändamål har många tekniker utvecklats såsom schemaläggning, sampling och värstafallet-analys. De har alla gett upphov till en stor mängd forskning med många viktiga tillämpningar, som alla blivit en oumbärlig del av utvecklingsprocessen.

Denna avhandling bidrar till valideringsfasen av utvecklingsprocessen med tyngdpunkt på valideringen av design och modellering på hög nivå. Inom det omfattande valideringsområdet fokuserar vi på en av de komplementära metoderna, nämligen *formell verifiering*. Denna metod använder datorkraft till att automatiskt fastställa graden av korrekthet i ett system med matematisk noggrannhet. För att kunna använda mekaniska processer för att fastställa delen av systemet huruvida systemet uppfyller de krav vi ställer, måste vi först modellera den validerade delen av systemet och kraven på korrekthet som matematiska objekt. Egenskaper hos det formella språket som vi använder för att beskriva det verifierade systemet och dess korrekthet är avgörande för komplexiteten hos verifieringsprocessen.

Denna avhandling behandlar en matematisk formalism som används som modell för realtidssystem, nämligen *tidsautomater*. Automatteori introducerar och studerar en av de matematiska modellerna som finns för beräkningar. Centrala koncept i denna modell är systemtillstånd och ändlig representation av övergångar mellan tillstånd. Ett tillstånd summerar all information om tidigare beräkningar som påverkar möjliga framtida utvecklingar av systemet. En ändlig beskrivning av övergångar mellan tillstånd bestämmer auto-

matens funktion. Baserat på tillståndsinformationen bestämmer övergångarna alla möjliga tillstånd som automaten kan drivas till.

Ett grundläggande exempel av en tillståndsbaserad modell, *en ändlig automat*, består av ändligt många tillstånd, och en övergångsrelation som explicit räknar upp alla möjliga övergångar mellan tillstånd. Strukturen hos en ändlig automat beskrivs ofta med en riktad graf med märkta kanter, noderna representerar tillstånd och kanterna övergångar. En ändlig automat accepterar uttryck som fås geom att länka samman märkningarna på kanterna längs en väg i grafen, givet att denna väg börjar i ett givet initialtillstånd och slutar i ett givet accepterande tillstånd.

Tidsautomater utvecklar denna grundläggande modell genom att lägga till en ändlig uppsättning reellvärda variabler (*klockor*), som automatiskt mäter förfluten tid. Varje övergång av en tidsautomat kan specificera lägre och övre gränser för tillåtna värden hos dessa variabler. Med tidsinformation i klockorna, och villkor för övergångar, lämpar sig tidsautomater som modeller för realtidssystem. Klockor och tidsvillkor gör det möjligt att modellera saker som deadlines, periodicitet eller generella tidsbegränsningar.

Fokus i denna avhandling ligger på automatiserad verifiering av realtidssystem modellerade som tidsautomater. Vi är intresserade av korrekthetsegenskaper som uppkommer i

- schemaläggning: Kommer alla deadlines att hållas?
- sampling: Bevarar samplingsfrekvensen systemets beteende?
- kontroll: Kan buffertar tömmas helt mellan kontrolluppgifter

Denna avhandling behandlar tidsautomater med avseende på bestämning av denna typ av egenskaper, framförallt logiska gränser för att automatiskt utföra verifieringen. Till exempel studeras i vilka sammanhang automatisk verifiering över huvud taget är möjlig.

En av hindren som måste överkommas för att syssla med automatisk verifiering, är att kunna hantera oändliga matematiska strukturer i modellen. Även om tillstånden och övergångsrelationen kan beskrivas ändligt, kan antalet tillstånd som behöver undersökas för en viss egenskap vara oändligt. Utan att finna och ta vara på speciella mönster i denna oändliga mängd tillstånd, skulle all automatisk verifiering vara omöjlig. I vissa fall, kan dessa mönster användas för att etablera metoder för formell verifiering som kan utföras av en dator och alltid ge ett korrekt svar efter ett ändligt antal steg.

Det finns två olika angreppssätt som öppnar dörren för oändliga strukturer i tidsautomater. En av möjligheterna är att utöka tidsautomater med ytterligare oändliga datastrukturer: obegränsade köer. Dessa köer kan fylla olika syften. Antingen kan de innehålla beräkningar och tillsammans med tidsautomaten modellera ett realtidssystem med olika uppgifter. Det intressanta problemet i denna kontext är schemaläggningen. Vi studerar här avgörbarhet och hur denna påverkas av egenskaper såsom preemption, varierande beräkningstid för olika uppgifter, och kommunikation mellan tidsautomaten och kön. Kön kan också användas för asynkron kommunikation mellan tidsautomater som synkroniserat körs parallellt. Dessa köer innehåller meddelanden från en au-

tomat och väntar på att läsas från av en annan automat. Sådana situationer uppkommer bland annat i realtidsreglersystem där ett antal parallella processer kommunicerar via buffertar. Vi studerar avgörbarheten för nåbarhetsanalys under olika topologier.

Den andra möjligheten är att studera en speciell egenskap hos tidsautomater som gör det möjligt att konstruera dem så att tidsskillnaden mellan händelser växer monotont men samtidigt är begränsad av en godtyckligt heltal. Detta kan karakteriseras av obegränsade räknare som håller reda på sådana tillväxter (ökningar). Om klockorna går i tät tid, spelar dessa ökningarna igen roll för de flesta verifieringsfrågor. När vi går över till en implementation som använder en fast klockfrekvens (samplad semantik), bevaras endast beteende som svarar mot en begränsad användning av dessa räknare. Vi beskriver dessa räknare som en ny typ av registermaskin och bevisar att man effektivt kan avgöra om räknarna används på ett begränsat sätt. Detta gör att man kan avgöra om det för en given tidsautomat finns en implementation med fast samplingsfrekvens som bevarar alla kvalitativa beteenden.

Contents

1	Introduction	13
2	Technical Background	21
2.1	Real Time Scheduling	21
2.2	Sampling of Dense Time	23
2.3	Channel Systems	26
3	Results	29
3.1	Task Automata	29
3.2	Sampled Semantics of Timed Automata	31
3.3	Communicating Timed Automata	32
4	Conclusions and Future Work	35
5	References	39
	Paper I: Task Automata	47
1	Introduction	48
2	Timed Automata with Tasks	50
2.1	Tasks and Scheduling Strategy	50
2.2	Task Automata	52
2.3	Operational Semantics	53
3	Schedulability Analysis	55
3.1	Schedulability of Task Automata	55
3.2	Decidability and Undecidability Results	57
4	Decidability	59
4.1	Timed Automata with Subtraction	59
4.2	Preemptive Schedulers as TA with Subtraction	61
4.3	Task Automata without Task Feedback	68
5	Undecidability	70
6	Conclusions and Related Work	74
	References	76
7	Appendix	78
	Paper II: R-automata	83
1	Introduction	83
2	Preliminaries	85
3	Universality	88
3.1	Concepts and Proof Overview	88
3.2	Construction of the Reduced Factorization Tree	93
3.3	Correctness	97
3.4	Algorithm	100
4	Limitedness	102

5	Büchi Universality	102
6	Non-emptiness	104
7	Conclusions	104
	References	105
	Paper III: Sampled Semantics of Timed Automata	109
1	Introduction	109
2	Preliminaries	112
2.1	Notation for Clock Differences and Regions	114
3	Results	115
4	Extended R-automata	117
4.1	Extensions of R-automata	117
4.2	Limitedness of Extended R-automata – Copy Operations	121
4.3	Limiting Maxima in Extended R-automata	127
5	Encoding of Timed Automata to Extended R-automata	129
6	Decidability Proof	139
6.1	ω -sampling	140
7	Conclusions	141
	References	142
	Paper IV: Communicating Timed Automata	147
1	Introduction	147
2	Communicating Timed Automata	149
3	CTA with One Channel	151
4	CTA with Two Channels	164
5	Conclusions	167
	References	169

1. Introduction

Nine-tenths of wisdom consists in being wise in time.
THEODORE ROOSEVELT

The successful applications of computers in the past, leading to their broad employment, makes us increasingly dependent on their success in various tasks we delegate to them every day. Let us view computers, including computing power embedded into other devices, as machines which calculate answers to our questions. Here are some examples of such questions:

- What is a feasible schedule for the next university term?
- What is the optimal pressure with which a car should break in a situation identified by the car sensor readings?
- On which runway should an arriving plane land?
- What is the next video frame to be displayed on the screen?

In all these cases, the computation succeeds if it delivers a correct answer in time. Timeliness of the result means that the question is still relevant at the timepoint when we know the answer. There are many questions, exemplified by our first example, for which the deadline is either flexible or far enough to leave space for a tradeoff between speed, cost, and other possible factors. The timing requirements are usually not stated explicitly in such cases and they can be summarized as "*the faster the better*".

The other examples require much faster response times. The sensor view of the car situation changes in order of milliseconds and the responsiveness of the control computer should correspond to the reaction time of a human driver. Decisions about allocating airport runways have to come in order of minutes taking into account both a pre-designed schedule and unexpected events caused by, e.g., weather or human interaction. The last example requires a refresh frequency sufficient to create an illusion of a smooth movement for human viewers. In all these cases, timing constraints given as, e.g., concrete bounds on response times or values of deadlines become an explicit part of the setting. If this is the case then we are dealing with *real time systems*.

For the second and the third question, a single failure to deliver an answer in time might have serious unwanted consequences. A violation of the constraints is not acceptable under any circumstances. In contrast to this, a missing frame in the last example causes a drop in the perceived quality of picture which, if not occurring too often, can be tolerated for most applications. The constraints could be then weakened by conditions like one out of five consecutive task instances can miss its deadline or at least 90% of deadlines have to be

met. In this thesis we deal with systems containing the first type of constraints – *hard* real time systems.

The fact that timing constraints become an explicit part of the setting gives rise to new problems, approaches, and solutions in the design of real-time systems. On the one hand, timing constraints add to the complexity of the system design. On the other hand, explicit statement of these constraints allows us to employ specific methods and tools. To this end, many concepts and areas have been established, e.g., scheduling, sampling, worst case execution time analysis. Each of them has spawned a huge body of research with numerous applications, which have become an indispensable part of the development process.

Development of real-time systems consists, as for other computer systems, of problem analysis, design of a solution, implementation, and validation. These phases are not strictly separate, they might more or less overlap, interfere, repeat at different places during the development. This thesis contributes to the validation phase of the development process with the emphasis on the validation of high-level designs and models. Within the broad area of validation, we focus on one of the complementary methods, namely *formal verification*. This method applies computer power to automatically establish correctness of the system model with mathematical rigor.

The unreachable (or even false) ideal of formal verification is to check whether the produced system functions correctly under all circumstances. It is not possible to achieve this goal for three reasons. In order to be able to apply formal verification

- the system has to be specified as a mathematical object; the produced system is often a physical object (or an executable),
- the system correctness has to be specified in a mathematical form; this is mostly not possible, we can express only some partial aspects of correctness, and
- we need to model the environment (all circumstances) in mathematical terms as well, e.g., as a set of possible input values.

Therefore, we first have to model the validated part of the system together with the environment in which it is supposed to function and the requirements as mathematical objects. Only then is it possible to apply mechanical procedures to determine whether the system satisfies the requirements. In spite of these restrictions, formal verification helps to discover corner case bugs, increase understanding of the problem, and remove ambiguities in the specification. Properties of the formal languages which we use to describe the verified systems and their correctness strongly determine not only the extent to which we have to abstract away from the details of the system, but also the complexity of the automated verification procedure.

This thesis deals with a mathematical formalism serving as a model of real-time systems – *timed automata*. Automata theory introduces and studies one of the mathematical models of computation. A central concept of this model is the explicit notion of a system (automaton) state and a finite representation of

the transitions between the states. A state of an automaton summarizes all information about the past computation which affects possible future evolutions of the system. A finite description of transitions between the states determines the operational possibilities of an automaton. Based on the state information, transitions prescribe all possible actions, which in turn might change the automaton state.

A basic instance of a state-based model, a *finite automaton*, consists of finitely many reachable states and a transition relation which explicitly enumerates all possible transitions. The structure of a finite automaton is often represented by a finite directed graph with edges labeled by letters, where nodes correspond to states and edges to transitions. A finite automaton accepts words obtained by concatenating the edge labels along paths in its corresponding graph, provided that these paths start in a designated initial node and satisfy a given accepting condition, for example, end up in one of the designated accepting nodes.

Since timed automata occupy a prominent position in this work, we describe this model in more detail in the next paragraphs. After this, we discuss their basic properties, the background in which they were developed, and their impact on the area of real time systems.

Timed automata were introduced in the seminal papers [6, 7] as an extension of finite automata. Syntactically, we add a finite set of real valued variables (*clocks*), which automatically measure passage of time. Each transition of a timed automaton can specify lower and upper bounds on time delays in terms of these variables. These constraints, called *guards*, compare individual clock variables to natural numbers, i.e., they are conjunctions of inequalities of the form $x \bowtie c$, where x is a clock, c is a natural number (including zero), and \bowtie is one of the relations $<, \leq, \geq, >$. A transition also specifies a (possibly empty) set of clocks which are *reset* to zero. Figure 1.1 depicts a sample timed automaton.

Semantically, a timed automaton can at each timepoint choose from two types of moves. It can delay in the current state for some amount of time. The values of all clocks then increase by this amount. Alternatively, it can instantaneously take a transition, provided that the clock values satisfy the transition guard. This transition also resets the indicated clocks as its side-effect. Clocks assume values from a given time domain. Semantic states are then pairs (l, ν) , where l is a state of timed automaton and ν is a function mapping clocks to values from the time domain. From now on, we distinguish the states of an automaton and the semantic states by calling the former *locations* and the latter *states*.

For example, if the automaton in Figure 1.1 is in the location l_1 and the values of the clocks x, y are 1.7, 0.93, respectively, then it can only delay, since the clock valuation does not satisfy the guard on any outgoing transition. After waiting for 0.3 time units, the guard $x \geq 2$ becomes enabled and the automaton can either take the corresponding transition or decide to keep waiting. If it takes this transition, it moves to the location l_2 and resets the clock y to 0.

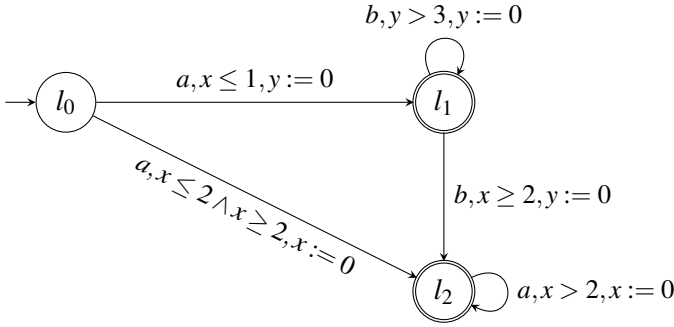


Figure 1.1: An example timed automaton with states l_0, l_1, l_2 and clocks x, y . The state l_0 is initial and the states l_1, l_2 are accepting. The labels on transitions denote the letter, the clock guard, and the reset. For instance, the transition from l_0 to l_2 is labeled by $a, x \leq 2 \wedge x \geq 2, x := 0$, where a is the letter, the constraint $x \leq 2 \wedge x \geq 2$ is the guard (often abbreviated by $x = 2$), and the clock x is reset along this transition.

Timed automata accept *timed words*, i.e., words where each letter is accompanied by a *timestamp* – a reading of the (global) time at which the corresponding transition was taken. An example of a timed word is $(a, 0.4)(b, 0.972)(a, 5.13)(c, 6.74)$, where a, b, c are letters and the numbers 0.4, 0.972, 5.13, 6.74 are timestamps. A timed word is either accepted or rejected depending on whether the transition sequence (*a run*) satisfies a given accepting condition. The most common accepting conditions are either reaching an accepting location at the end of the run [37] in case of finite words or standard Büchi acceptance conditions [7] for infinite words. For example, the timed automaton in Figure 1.1 accepts the finite timed word $(a, 0.4)(b, 1.756)(a, 2.1)(a, 4.2)$.

Sometimes, we are interested only in qualitative behaviors of a timed automaton – we omit the timing information. This corresponds to the untimed language consisting of letter sequences, i.e., the set of words obtained from timed words by projecting out the timestamps.

The most commonly considered time domain is the set of non-negative real numbers. It is also used in this thesis and the semantics is then referred to as *dense time semantics*. Replacing real numbers by the set of non-negative rational numbers would allow for a finite representation of each semantics state, which means that the set of states would be countable, while retaining density of time. Real and rational semantics differ only in very subtle points and none of the results presented here would change with rational numbers as the time domain. When we restrict time delays to multiples of a fixed rational number, we talk about *sampled semantics*. The reduction of dense time behaviors to discrete ones by choosing an appropriate value of the smallest time step is called *sampling*.

With timing information in clocks and timing constraints on the transitions between the states, timed automata serve as a model for real-time systems.

Clocks and timing constraints allow us to model many concepts involving explicit timing information, such as deadlines, periods, timeouts, or interval time bounds in general.

The main focus of this work is on automated verification of real-time systems modeled by timed automata.¹ We are interested in correctness properties from

- scheduling: Will all deadlines be met?
- sampling: Are there sampling rates preserving all qualitative behaviors?
- control: Can buffer underflows occur between control tasks?

This thesis studies timed automata with respect to determining validity of such properties, mainly the logical limits of performing the analysis automatically, i.e., determining in which settings formal verification is or is not possible for fundamental logical reasons.

One of the obstacles to a successful automated verification are infinite structures in the mathematical model. Even if the states and the transition relation admit a finite representation, the number of states which need to be analyzed in order to establish validity of the correctness properties might be infinite. Without discovering and utilizing specific patterns in this infinite amount of states, any automated verification would be impossible. In some cases, one can make use of these patterns and devise procedures for formal verification which can be executed by computers and always give a correct answer after finitely many steps.

The most prominent source of infinity in timed automata is the set of clock valuations, which is even uncountable when considering real numbers as the time domain. Therefore, the set of (semantics) states is infinite or even uncountable. This fact leads to most of the negative results concerning automata properties, formal verification of timed automata in particular. If we consider timed languages then the alphabet is infinite. Also, the automata can exhibit unrealistic so called Zeno-behaviors when there are infinitely many discrete events happening within a bounded amount of time. The other source of infinity considered in this work is unbounded queues added as an extension to timed automata.

One of the most important properties of timed automata, exemplifying a successful analysis of infinite state spaces, is decidability of the language emptiness problem [7]. The decision procedure groups the reachable states into finitely many finitely represented classes and visits each of them at most once. All states within one class can be inspected by analyzing a finite representation of this class.

On the negative side, the language universality problem is undecidable for timed automata and as a consequence, also the language inclusion problem is undecidable. This means that timed automata cannot serve as both the correctness specification language and the system description language when we

¹Other aims of such remodeling of problems from established areas by timed automata might be a new view on these areas and a better understanding of modeling capabilities and limits of timed automata.

want to verify automatically that a system satisfies its specification. The class of languages accepted by timed automata is also not closed under complement, which has strong consequences for instance for logical characterizations of this class.

Timed automata were introduced in a period when the research community developed several other formalisms for real time systems, both from within the automata theory, based on the notion of state, and from outside the automata theory. The most prominent state based formalisms of this time are a timed version of I/O Automata (referred to as Timed I/O Automata here) [49], Timed Transition Systems [35], Modechart [39], and Timed Petri Nets [51, 14]. Other formalisms for specification of timed systems include process algebras [56, 61, 53] and logics, e.g., Duration Calculus [21, 20], Metric Temporal Logic [41], Metric Interval Temporal Logic [8], Timed Propositional Temporal Logic [9] and Timed Computational Tree Logic [5]. In the following paragraphs, we present closer the state based formalisms.

Timed I/O Automata operate in dense (real) time. Timing is added to I/O Automata by "external" timing conditions, i.e., the model is a pair of an I/O automaton and a set of timing conditions. These conditions specify minimal and maximal time which can elapse after a given set of states is entered (or a transition from a given set of transitions is executed) before a given event is observed (or a given set of states is reached).

Modechart was developed as a specification language with semantics given by a translation to Real Time Logic [38]. This discrete time model consists of a hierarchical structure of modes (which resemble states in automata models). Timing information is incorporated into conditions on the transitions and expressed by Real Time Logic predicates. These constraints specify minimal and maximal time which the system can spend in the current mode after entering it.

Timed Transition Systems form a discrete time computational model. Transitions between states (which are valuations of system variables) are constrained by a lower and upper bound on the time continuously spent in this state. This means that time is always measured from the moment when the system enters a state.

Timed Petri Nets extend the popular model of Petri Nets by timing constraints and dense time semantics. Each transition is equipped with two rational numbers, a lower bound and an upper bound. A transition can be fired if it has been continuously enabled for at least as many time units as given by the lower bound and it has to be fired before it has been continuously enabled for the upper bound time units.

The most significant distinguishing feature for timed automata is the explicit handling of clocks. The fact that clocks automatically measure passage of time together with the ability to branch upon the clock values and to reset clocks to zero allows us to model the timing properties locally and in an imperative way. The automaton resets clocks and measures their values whenever it is needed (and desirable) without explicitly considering all possible future or past behaviors.

Timed automata induced a considerable body of research. There has been various verification tools (model checkers) developed for timed automata, the most prominent being Kronos [62] and Uppaal [46]. These tools contributed to the research of techniques improving scalability of model checking and helped in popularization of formal modeling and analysis in the university education. Timed automata also contributed to research in other areas, such as logics [59, 24], hybrid systems (timed automata are a clear simple case of hybrid systems, see e.g., [34]), and scheduling (modeling of schedules by timed automata [1], modeling of release patterns by timed automata [25]). Also, many well-established research areas covered real time systems by extending their concepts for timed automata, exemplified by game theory [23] and learning [32].

Now we have introduced both the general topic of this thesis, which is the formal verification of real time systems, and the mathematical model on which we focus our attention – timed automata. Before we present our results, it will be necessary to describe the areas of our research in more detail.

2. Technical Background

Be not curious in unnecessary matters: for more things are shrewd unto thee than men understand.
Sir 3:23

This chapter introduces the research areas of this thesis and motivates its contribution. The first two areas, scheduling and sampling, are well-established within the real-time systems research. The third area, channel systems, has been mostly studied in connection with communication protocols. Motivated by real-time control, we combine channel systems with explicit timing and investigate their properties.

2.1 Real Time Scheduling

Scheduling problems occur when several agents compete for a shared limited resource. Such limited resources can be processor time, I/O bus time, time slots on a given radio frequency, but also time for which an expensive tool such as a microscope, an operation theater, or a lecture room can be used. Agents, e.g., computation tasks, phone calls, biologists, patients, or teachers need an exclusive access to the corresponding resources in order to complete a computation task, a phone call, a scientific experiment, an operation, or a lecture. The area of scheduling deals with the problem of allocating the shared resources to the agents while satisfying given criteria. These criteria usually talk about the time needed to finish the jobs, but they can also take into account other aspects such as quality of service, or costs.

The most general problem can be stated as a search for a description of a resource allocation over time, a *schedule*, such that all given criteria are satisfied. As an example, consider creating a schedule for a university department, where all lecturers have certain requirements on the lecture rooms, lab rooms with special equipment, lecture times, and so on. Such a schedule is usually static, i.e., completely known in advance. There can also be dynamic schedules, so called *scheduling strategies*, which describe a procedure deciding who should occupy which resource under given circumstances. An example of such a dynamic schedule is a priority assignment, e.g., the most seriously injured patients are operated first while the others wait. For an overview of scheduling algorithms and further references, see [17, 48].

A simpler yet important problem is to check whether the system satisfies all criteria with a given scheduling strategy – the *schedulability analysis* problem. For a dynamic schedule, one has to check that all possible scenarios

which might occur during system's operation satisfy the criteria. When a given scheduling strategy fails in this test or when there is no scheduling strategy given, we can also ask whether there is a feasible one at all.

To formulate the schedulability analysis problem in a mathematical form, one has to consider an abstraction of the whole scheduling setting. In the context of computer systems, the abstraction often consists of a set of computation tasks, a processor (or several processors), task release patterns, a task queue, and a scheduling policy in question. The computation tasks are identified by parameters like a computation time C and a deadline D . Each task τ represents a piece of executable code which needs C_τ time units of the processor time to be finished and it should finish earlier than D_τ time units after its release. Processors offer a uniform (invariable in time) source of computing power. The tasks arrive in time according to a given arrival pattern and are inserted into the task queue at the moment of their arrival, where they wait to be computed. The scheduling policy decides which task should execute at each timepoint.

In order to analyze schedulability of recurring tasks with hard deadlines, it is common to consider arrival patterns to be *periodic* [48]. The periods then become additional task parameters. To relax from this rather restrictive scheme, the imprecision of task arrival times is often modeled by a *jitter* – a bound on the allowed deviations from the strictly periodic pattern. Classical results [47] in this setting include the utilization bound for schedulability of tasks scheduled according to fixed priorities and the optimality of the Earliest Deadline First scheduling policy for a single processor.

The release patterns characterized by periods and jitters often constitute a reasonable abstraction of real time systems. However, they do not contain any state information, which makes modeling of, e.g., sporadic tasks or different modes of operation difficult. To overcome this limitation, timed automata have been suggested as a modeling language for release patterns in [25]. Locations of a timed automaton contain task identifiers and each time a location is entered, the corresponding tasks are released. The runs of the automaton then correspond to concrete evolutions of the system. By this, it is possible to model sporadic events, different modes of operation, and to incorporate the influence which the past evolution of a system has on its future (through the information summarized in states).

The following work [29, 44, 28] analyzed the theoretical properties of this extension. Scheduling policies considered here are either Fixed Priority Scheduling or Earliest Deadline First and the focus is on decidability of the schedulability analysis problem for systems modeled by timed automata with tasks. The decidability results lead to a prototype implementation and an experimental evaluation of this approach [11, 12, 27].

We work with a model which incorporates different features from scheduling, namely, preemption, uncertainty in the task execution times, and data dependencies into the timed automata setting. By this we obtain a strong modeling formalism with precise mathematical semantics, which should ease formal modeling for the scheduling community. In the first place, we are concerned with expressiveness of this new model in terms of basic automata theory.

One direction of research aims at establishing the decidability borderline of the reachability problem for the timed automata based model enriched by various scheduling features. The infinite structures in this model comprise the clock valuations induced by timed automata and an unbounded queue for storing the released tasks. Positive answers to the decidability question essentially mean that the extended model has a finite representation equivalent with respect to schedulability. This enables application of standard automata based formal verification techniques directly to the scheduling problems modeled in our setting, especially to the schedulability analysis problem.

In order to analyze this model, one has to formalize the class of admissible scheduling policies. The goal is to establish the properties which a scheduling policy has to satisfy to be suitable for automatic schedulability analysis. We identify the properties which guarantee that it is sufficient to consider only task queues of a bounded size and that the density of time cannot be used to encode and manipulate natural numbers.

Timed automata (or their extensions) have been used in the scheduling setting also in the following works. Synthesis of dynamic state-based schedulers has been developed in [3]. The synthesized scheduler is represented by a timed automaton whose runs describe all correct schedules. In [1, 2], runs of timed and stopwatch automata model feasible schedules for a job-shop scheduling problem and an algorithm is presented for finding a time optimal one. Other models, a timed process algebra [45] and Real Time Logic [52], have also been used to synthesize schedules for hard real time systems.

A similar way of encoding systems with tasks in the timed automata setting has been developed in [50], where the authors identify a subclass of hybrid automata for which the language emptiness problem is decidable. They demonstrate the usefulness of this subclass by modeling the Shortest Job First scheduling policy.

2.2 Sampling of Dense Time

Dense time semantics allows each delay taken by a timed automaton to be an arbitrary non-negative real number. This includes also arbitrarily small delays and delays which differ from each other by arbitrarily small values. Even though the other type of behaviors might occur as a result of impreciseness such as clock drifts, neither of these behaviors can be enforced by an implementation operating on a concrete hardware. Each such an implementation necessarily includes some (hardware) digital clock which determines the least time delay measurable or enforceable by the system.

This observation motivates sampled semantics of timed automata, which is a discrete time semantics with the smallest time step fixed to some fraction of 1. In other words, the time delays in a sampled semantics with the smallest step ε can be only multiples of ε . There are infinitely many different sampled semantics, but any of them allows fewer behaviors of the system than dense time semantics. On the other hand, all of the allowed behaviors in a sampled

semantics with the smallest time step ε will be preserved in an implementation on a platform with the clock rate ε (and all fractions of ε).

To characterize the relation between dense time semantics (giving us a set of dense timed words) and sampled semantics (giving us a set of sampled timed words), we restrict ourselves to qualitative behaviors of the automata. This means that we consider only untimed words. The untimed word obtained from a timed word $w = (a_1, r_1) \dots (a_k, r_k)$ is the sequence of letters $a_1 \dots a_k$.

By considering only qualitative behaviors (untimed languages) we lose the explicit timing information, but many important properties, including implicit timing, are preserved. For instance, if we know that the letter b cannot appear later than 5 time units after an occurrence of the letter a in the dense time model and then there is an untimed word accepted by this automaton where a is followed by b then we know that there is a run where b comes within 5 time units after a .

The goal of studying sampled semantics of timed automata is to contribute to the discussion about using dense and discrete time in formal verification. Dense time semantics provides us with a very convenient feature. In the modeling and verification phase, we do not have to consider a concrete sampling rate of the implementation. Clock comparisons do not depend on it as we assume that all constants are multiples of any admissible smallest time step. One can see dense time semantics as including all (infinitely many) sampled semantics and arbitrarily choosing a sampling rate before every delay move of the timed automaton.

We investigate the relation between dense time semantics and discrete time semantics by studying whether and in which way can the dense time model exhibit behaviors without a corresponding counterpart in sampled semantics. By this we should be able to detect whether all qualitative system behaviors observed in dense time are preserved by some implementation with a suitably chosen fixed sampling rate or whether some behaviors will be lost regardless of the clock frequency.

We consider the following formulation of the sampling problem: decide whether there is a smallest time step ε such that the untimed language accepted by a given timed automaton is the same in both the dense time semantics and the sampled semantics with ε . If we consider untimed words as an identification of qualitative behaviors then the problem could be rephrased as follows: is there an ε such that all qualitative behaviors of a given timed automaton in the dense time semantics are preserved by an implementation with the time step ε ?

It has been observed before [13, 26] that there are timed automata which cannot be sampled without losing untimed behaviors. This observation relies on a peculiar property of the dense time semantics of timed automata – the ability of enforcing behaviors where time distances between events monotonically grow while being bounded by some integer. The cause of such behaviors is the ability of timed automata to specify that an event happens strictly earlier or later than a timepoint marking an integral distance from some other event. This is possible because of the strict inequalities $<$ and $>$ in the automata

guards. Closed timed automata, i.e., timed automata with only \leq, \geq inequalities in the clock guards, can be always sampled with the sampling rate 1. Closed timed automata possess one important property. They are *closed under digitization* ([54], Property 8). This property has been defined in [36] and it is connected to our problem in the following sense: if the timed language of a timed automaton is closed under digitization then all (untimed) behaviors of this timed automaton are preserved with ε equal to 1.

Studying this problem can be seen as an investigation of the nature of the behavior loss during sampling caused by strict inequalities. Timed automata can enforce growth of the distances between the fractional parts of clock values. This corresponds to a special way of counting, namely counting how many times did some distance grow in the current run. Timed automata then possess some type of unbounded counters, which cannot influence the decisions in the computation, but which can block a computation in presence of sampling. The bigger the counter values are, the finer sampling rate one needs to preserve the runs. Characteristics of these counters closely resembles and extends distance automata of Hashiguchi [33] and newer work [58, 40, 15, 22].

The problem of asking for a sampling rate which satisfies given desirable properties has been studied in [13, 26]. Both of these works also observe that there are timed automata (with strict inequalities in guards) for which there is no sampling rate preserving all qualitative behaviors. In [13], the authors identify subclasses of timed automata (or, digital circuits which can be translated to timed automata) such that there is always an ε which preserves qualitative behaviors. The problem of deciding whether there is a sampling rate which would ensure the language non-emptiness is studied in [26] with a negative answer (Theorem 3). However, the definition of sampled semantics, motivated by the control setting, differs from that of ours in that the automaton is obliged to take a discrete transition after every time tick. The undecidability proof makes full use of this difference. Based on this work, the problem of determining whether there is a sampling rate such that the infinite word language of a given timed automaton is non-empty with our definition of sampled semantics has been wrongly classified as undecidable in [10]. Later work [43] presented a decision procedure for this problem.

Digitization of timed languages has been studied in [36]. This work identifies properties of systems and specifications which allow us to transfer the verification results obtained for the discrete time to the dense time setting. Digitization takes into account timing properties more explicitly, while in our setting the stress is on the qualitative behaviors (untimed language). Another different approach to discretization has been developed in [31]. The discretization scheme suggested there preserves all qualitative behaviors for the price of skewing the time passage.

Implementability of systems modeled by timed automata on a digital hardware has been studied in [60, 42, 4]. The papers [60, 42] propose a new semantics of timed automata with which one can implement a given system on a sufficiently fast platform. On the other hand, [4] suggests a methodology in which the hardware platform is modeled by timed automata in order to al-

low checking whether the system satisfies the required properties on the given platform.

2.3 Channel Systems

Channel systems, sometimes also referred to as *communicating finite state machines*, essentially consist of a finite set of finite automata (indexed by natural numbers) with a special alphabet. Letters consist of three parts: an automaton index, a symbol distinguishing sending and receiving operations, and a message chosen out of a finite set of possible messages. We adopt the standard notation demonstrated by the letters $4!a$ and $3?b$, which say that message a is sent to the automaton with index 4 and that message b is received from the automaton with index 3, respectively. There is also an additional letter ε in the alphabet, which denotes that the automaton neither sends nor reads any message.

The operation of these machines assumes that the automata are connected by unbounded directed *channels* following the first-in-first-out policy. Messages sent by one automaton to another one are inserted into the corresponding channel, where they stay until they are read. An automaton can read messages which are at the head of incoming channels by taking a transition labeled by the corresponding letter. If a label on some transition does not match the channel content then this transition is blocked. In this respect, the content of channels partially¹ determines the automata behavior.

Channels are assumed to be perfect, i.e., there are no losses, insertions, or modifications. When we say, e.g., that a channel system has one channel we mean that only one channel is used. We assume that there can also be channels from an automaton to itself, e.g., a channel system with one automaton whose transitions are labeled only by $1!a$ and $1?a$ for all messages a . Given two automata A_1, A_2 , we denote the channels between them by $c_{1,2}$ and $c_{2,1}$. To specify a channel system, we list the automata and the channels in one tuple. For example, $(A_1, A_2, A_3, c_{1,1}, c_{1,3}, c_{2,3})$ describes a system with three automata, where the first one has a channel leading back to itself and both the first and the second one have a channel for sending messages to the third one.

There are many properties of channel machines which one would like to check automatically in order to verify the correctness of the system behavior. Here we list some of them.

- *Reachability* – is there a run of a given system which reaches a concrete control state together with given channel contents?
- *Deadlock* – is there a reachable state where all automata can just receive and all channels are empty?
- *Boundedness* – is the set of all reachable states finite?

The first work [16] studying decidability of these problems showed that the general model is Turing complete, by showing that systems of the form

¹It cannot resolve non-determinism in choosing writing operations and ε transitions.

$(A_1, c_{1,1})$ can simulate computations of a given Turing machine. Intuitively, the channel contains a tape content together with a machine control state. The automaton A_1 reads the tape content in windows of finite size from the channel. According to the window and the control unit of the Turing machine it computes the next window and directly writes it symbol by symbol back to the channel. As a consequence, none of the problems mentioned above is decidable for general channel systems.

Further work focused on subclasses of channel machines and strengthened the undecidability results. Two identical “daisy” automata with one channel in each direction can simulate a given Turing machine [30]. Surprisingly, also two automata connected by two channels going in the same direction can simulate Initial Post’s Correspondence Problem (reachability of a given control state with empty channels can be reduced to this problem [55]). Intuitively, to simulate a Turing machine on such a system, one sends configurations of a Turing machine computation into both of them. The computations are shifted by one configuration, so that the receiving machine can check whether all moves have been done correctly.

To sum up the undecidability results, two channels are enough to simulate a Turing machine and one channel suffices under the condition that the receiver can pass a finite amount of information to the sender, e.g., by using shared states (in the extreme case, when the sender and the receiver are the same automaton).

To present the decidability results, we need two more technical definitions. A channel machine $(A_1, A_2, c_{1,2}, c_{2,1})$ is *half-duplex* if in all reachable states at most one channel is nonempty. This property is already mentioned in [55], but further studied in [18]. A channel machine is *cyclic* if it is of the form $(A_1, \dots, A_n, c_{1,2}, c_{2,3}, \dots, c_{n,1})$.

Stability is decidable for cyclic channel machines with one of the channels bounded [55]. It follows from the fact (often used in other decidability results) that for a cyclic channel machine, one can rearrange each infinite computation so that only one arbitrarily chosen channel is not bounded by 1. To achieve this, we need to assign priorities to the finite automata in an appropriate way.

In [18], the authors show that reachability, boundedness, and other properties are decidable for half-duplex systems. Moreover, it is decidable for a channel system whether it is half-duplex. Since systems of the form $(A_1, A_2, c_{1,2})$ are a special case of half-duplex systems, reachability and boundedness are decidable for them. If we talk about a language accepted by such a system, e.g., words written into the channel, then this language is regular.

All of the above mentioned works assume that the automata work asynchronously. This assumption comes from the domain of interest – communication protocols. Protocols for communication between machines with high computation speed connected by channels with long and unpredictable communication times can often be faithfully modeled by an asynchronous system. However, for real time control systems with several processes communicating through buffers, the asynchrony assumption is not plausible, because the buffers can deliver messages almost instantly and the processes do not idle

when having relevant input. Also, both timing of process computations as well as communication patterns play a crucial role in the system correctness.

Therefore, we study channel systems in the synchronous setting. One possibility to bring in synchrony is to require that all automata take transitions simultaneously and in the same pace. This gives rise to discrete time automata. One more step leads to the dense time, where the communicating machines are timed automata. For this model, we obtain the synchrony for free, because the clocks evolve by definition in the same pace for all automata involved in the system.

An objective of this research is to develop a model for real-time systems communicating through channels such that one does not have to specify any bound on the channel size at the modeling and verification time. We propose to combine timed automata and channel machines in order to achieve this goal. By this we bring together two concepts which often render automated analysis logically impossible – unbounded queues and dense time. Each of these features contributes to the infinite amount of reachable states. The state space contains all possible message sequences in the channels and all clock valuations. We investigate possibilities and limits of verification of timed automata communicating through unbounded channels, which is also, according to our best knowledge, the first attempt to study channel systems with unbounded channels in the synchronous setting. This research also brings insight into what amount of information can be implicitly exchanged between machines by a common knowledge of global time.

Synchronous channel machines were studied in [19], where each channel cell is a finite automaton with a transition function taking into account also the content of the cell on the left. Such systems can recognize languages accepted by linear time bounded alternating Turing machines.

3. Results

People love chopping wood. In this activity one immediately sees results.

ALBERT EINSTEIN

Contributions of this thesis span over all three areas described above. First, we analyze possibilities of modeling various scheduling concepts using timed automata. Next, we study the relation of dense and discrete time in transition from dense time to sampled semantics of timed automata. The goal is to determine a sampling rate preserving all qualitative behaviors or present a witness of the fact that there is no such sampling rate. Finally, we investigate expressive power of a model combining synchronization by dense time and communication through unbounded channels.

3.1 Task Automata

This work aims at better understanding of *task automata* – an automata based model for real time systems where tasks compete for a scarce resource. The contribution is twofold. First, we characterize a class of scheduling properties for which schedulability analysis is possible. Secondly, we show a limit of extending the setting with scheduling features by proving that a combination of three natural features modeled in task automata turns the system Turing complete.

A task automaton, introduced in [25], is a collection of task types and a timed automaton modeling task release patterns. Each task type is characterized by its name, the best case computing time B , the worst case computing time W , and a relative deadline D . The timed automaton in question contains an additional labeling function which assigns task types to locations.

To define behaviors of a task automaton, we need a mechanism which picks one of the waiting tasks to be executed – a task queue and a *scheduling policy*. We assume that the released tasks wait in a queue and the first one is executed. A scheduling policy takes care of newly arrived tasks. It is a function which takes a list of tasks (a task queue) and a task type as arguments and returns a task queue. There are important constraints on the scheduling functions we consider; we discuss them below.

We describe the semantics of a task automaton with a scheduling policy as follows. Whenever the automaton enters a location, an instance of the corresponding task type is released. At this moment, the scheduling policy updates the current task queue with the released task type (in fact, it inserts this task instance at some position in the task queue). The system contains one proces-

sor where the first task in the queue runs. A task instance may be removed from the queue when it has been processed for at least B time units and it has to be removed from the queue when it has been processed for W time units.

The *schedulability analysis* problem is to decide for a task automaton and a scheduling policy whether there is a run of this system where some task instance misses its deadline, i.e., it stays in the queue for more than D time units. If it is not the case, i.e., all tasks meet their deadlines, we say that the task automaton is schedulable with the particular scheduling policy. A task automaton is *schedulable* if there is a scheduling policy such that it is schedulable with this policy.

We identify three features of real time systems with tasks, used and modeled in task automata setting in [29, 44, 28].

- Preemption – scheduling policies can insert the new task instance at the head of the queue, preempting the currently running task.
- Variable execution time – some task types can have $B \neq W$, allowing different non-deterministically chosen computation times for different instances of the same task.
- Feedback – the task automaton contains a special clock which is reset each time a task finishes. By this, the automaton can adjust task releases depending on the task finishing times.

The first contribution of this thesis is a characterization of a class of scheduling policies such that the schedulability analysis problem becomes decidable for task automata with fixed computation times ($B = W$ for all task types; preemption and feedback are allowed) or without preemption (variable computation times and feedback are allowed). It remains an open question whether schedulability analysis with a given scheduling policy is decidable for variable computation times and preemptive scheduling policies when feedback is not allowed. The decidability proof reduces the schedulability analysis problem in several non-trivial steps to the reachability problem of timed automata.

The first condition on the scheduling policies required by our definition is that a scheduling policy only inserts new task instances at some position in the current task queue. Especially, it cannot change the order of, add, or remove other task instances. Moreover, this definition requires existence of a computable function which for each size of the task queue gives a timed automaton of a special form deciding at which position is the new task instance inserted.

Our second contribution shows that when the system uses all three features, i.e., preemption, variable execution times, and feedback, it can simulate a two counter machine. This holds for most of the reasonable scheduling policies, including Fixed Priority Scheduling, Earliest Deadline First, and Shortest Job First. The proof encodes counters of a two counter machine into clock values of the task automaton in a similar way as in [34]. Due to the high expressive power, there is no algorithm determining whether a system is schedulable with these scheduling policies.

3.2 Sampled Semantics of Timed Automata

Let $L(A)$ and $L_\varepsilon(A)$ denote the set of accepted untimed words (qualitative behaviors) of a timed automaton A in dense time semantics and sampled semantics with time step ε , respectively. The problem of our interest is whether for a given timed automaton A there is a sampling rate ε such that the corresponding sampled semantics contains all accepted qualitative behaviors, i.e., $L_\varepsilon(A) = L(A)$.

We call this problem the *sampling* problem and we show that it is decidable. The proof requires a new type of counter automata which gives an insight into the dense time behaviors of timed automata, but also constitutes a contribution in its own right.

First we point out an observation which makes this problem non-trivial: timed automata can enforce the clock value differences to increase or decrease while being strictly bounded by 1 or 0, respectively. Figure 3.1 shows such a timed automaton. If $0 < a < b < 1$ are the values of x, y in the location l_0 and c, d are the values of x, y when the automaton enters the location l_0 again after reading ab then $b - a < d - c$. If such a fragment is repeated n times, the clock difference has to increase n times. In the dense time semantics, the increases can be arbitrarily small and thus their sum can be bounded by 1. In a sampled semantics with the clock rate ε , the sum of the increases is greater than or equal to $n \cdot \varepsilon$.

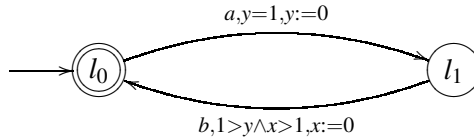


Figure 3.1: A timed automaton which does not preserve qualitative behaviors in sampled semantics. It enforces the difference between clock values to grow. If the values of x, y are 0.1, 0.6, respectively, in the location l_0 then the difference between the clock values in the location l_0 after reading ab will be strictly greater than 0.5. This example is adapted from [10].

The ability of timed automata to enforce growth or decrease of the clock differences (e.g., $b - a < d - c$ in the previous example) is not strong enough to specify how much should the clock difference grow or decrease. This feature is investigated and utilized in our result.

To solve the sampling problem, we have to analyze the effects on the clock differences along all possible accepting runs. If there is a sequence of (untimed) words for which accepting runs enforce more and more difference increases then there is no sampling rate preserving all of them. Figure 3.1 depicts such a timed automaton. Each string of the form $(ab)^k$ for some natural number k is accepted in the dense time semantics but for each ε there is a natural number k such that $(ab)^k$ is not accepted in the ε -sampled semantics. This is

because the loop enforces an increase in the difference between clocks x and y . At the beginning, the values of both clocks are equal (their difference is zero), but after the first loop (reading ab), the value of x is zero and the value of y is greater than zero (and smaller than 1). After each next loop, the difference between the clocks has to grow by a strictly positive amount, while both clocks stay smaller than 1 (in the initial location). Violating this requirement will lead to a deadlock, i.e., to a situation when no further letter may be read.

A timed automaton then possesses a special type of memory – a counter for each pair of clocks recording the information about how many times did the difference between the fractional parts of these clocks have to grow during the current run. To analyze the nature of this memory, we propose a new model of counter automata – *Extended R-Automata (ERA)*. An extended R-automaton is a finite automaton with a finite number of counters which assume natural numbers as their values. The operations on the counters include an increment, a reset to zero, a copy, and a restricted operation of taking maxima of the counter values. The property of interest for ERA is whether there is a bound such that all words accepted by ERA can also be accepted by a run along which no counter exceeds the bound. Deciding whether there is such a bound is known as the *limitedness* problem in the literature [33].

We show that the sampling problem for timed automata can be reduced to the limitedness problem for ERA. This reduction simulates each update of clock values by a corresponding operation on the counters as to record the number of the increases/decreases of the clock value differences. The correctness proof then implies that the available counter operations are sufficient to capture the effects which timed automata can have on clock value differences.

Decidability proof for the limitedness problem for ERA is split into several steps. First, we deal with a subclass of ERA, called *R-automata*, which can only increment or reset the counters. We present an algorithm deciding whether a given R-automaton is limited and prove its correctness. This proof utilizes a deep result from semigroup theory – *factorization forest theorem* [57]. In the next two steps we extend R-automata with the copy and maximum operations and reduce the limitedness problem for the extensions to limitedness of the original R-automata. The reduction for the operation of taking maxima requires special restrictions on when the maximum operation can be used. It is an open question whether limitedness is decidable for ERA with unrestricted maxima.

3.3 Communicating Timed Automata

Inspired by control systems where several real-time processes communicate via buffers, we define synchronous channel systems by assuming that the communicating machines (modeling the processes) are timed automata. The labels on the timed automata transitions encode sending or receiving messages to or from the unbounded channels or idle waiting (the label ε). Moreover, the transitions are labeled by clock guards and clock resets as for standard timed

automata. All clocks in the whole system run in the same pace, but each timed automaton can access (test and reset) only its own clocks.

Processes in real time control systems are designed not to stay idle in a situation when they have a relevant input. This is reflected by the following semantic feature. We require that timed automata read messages from the channels in an *urgent* manner – the reading automaton is not allowed to take an ϵ transition (which models staying idle) if there is a message in an incoming channel for which it has an enabled reading transition.

We study decidability of reachability problems for this model on two simple topologies. First, we investigate the system with two timed automata connected by one channel of the form $(A_1, A_2, c_{1,2})$. We show that the reachability problem is decidable. To be able to talk about the expressive power of these systems, we define the accepted language as the set of words constructed by concatenating the transmitted messages along runs which bring all automata to an accepting state. We show that the expressive power of this topology grows from regular languages in the asynchronous setting to one-counter languages in the synchronous (timed) setting. This holds true even if we consider this model with discrete time semantics. Without the urgency assumption, this model also accepts some non-regular languages.

The proof utilizes two techniques, reordering from [55] and clock difference relations from [43] to handle the dense time. Informally, each computation can be reordered so that there is always at most one message in the channel or the messages are not read anymore. In contrast to the asynchronous setting, this reordering requires desynchronization of the two timed automata. To keep track of the desynchronization, we need an integral counter and a special data structure – clock difference relations – to handle the dense time aspects.

Secondly, we show that the linear topology with three timed automata connected by two channels in one direction can simulate Turing machines. This result strongly contrasts with the asynchronous case, where such a topology accepts only regular languages. The urgency assumption is crucial here, but the expressive power does not depend on the density of time. In other words, such a topology can simulate Turing machines even with discrete time.

The fact that our results hold also for discrete time demonstrates that the unbounded channels add an independent degree of infinity and dense time does not increase the expressive power for timed channel systems.

4. Conclusions and Future Work

If you wish to advance into the infinite, explore the finite in all directions.
JOHANN WOLFGANG von GOETHE

This thesis investigates infinite structures which arise in mathematical models of real time systems. Understanding properties of these structures is crucial for designing procedures for automated verification of these models. The mathematical formalism we have studied – timed automata – operates in dense time. As a consequence, there are infinitely many reachable states for all non-trivial models. We have also introduced and investigated other sources of infinity, namely unbounded queues and sampling. Our work contributes to three different areas of real time systems.

Schedulability Analysis of Task Automata.

The theory of timed automata proves to be a promising tool for modeling and analysis of scheduling problems. It allows to specify release patterns of sporadic events or various modes of operation of a real-time system. It also brings all formal verification results and technology to be readily used. Earlier work [29, 28] has shown that schedulability analysis is decidable for task automata with the Earliest Deadline First and Fixed Priority scheduling policies.

As our first contribution, we formulate criteria on scheduling policies such that it is possible to perform automated schedulability analysis. These criteria are very general and accommodate many standard scheduling policies. The second contribution in the area of scheduling shows that a combination of very natural features (preemption, variable computation times, feedback at task finishing times) together with dense time makes task automata Turing complete. As a consequence, we have to give up hope for fully automated verification. On the positive side, we have determined interesting and non-trivial combinations of features which allow for schedulability analysis (both with and without a given scheduling policy). In short, we can decide whether there is a scheduling policy guaranteeing no deadline misses for a system if one of the above mentioned features is not used.

The remaining open problem in our setting is decidability of schedulability analysis for task automata with variable computation times and preemption, but no feedback from the task queue back to the automaton. Also, there have not been many results shown for the case of several processors, each having an independent task queue.

Sampling Dense Time Behaviors of Timed Automata.

We have investigated the problem of *sampling* dense time behaviors with a fixed sampling rate. As has been observed earlier [13, 26], there are timed automata for which no sampling rate exists to cover all qualitative (untimed) behaviors. We have shown that it is decidable whether a given timed automaton can be sampled while preserving all qualitative behaviors.

Timed automata with dense time semantics can enforce behaviors where time distances between events monotonically grow while being bounded by some integer. Such behaviors fully exploit density of time. Reachability analysis, which utilizes the region abstraction for timed automata state space obscures this fact by disregarding any information related to runs. In the first step, we have characterized the ability of enforcing these behaviors by a new type of counter automata – Extended R-automata (ERA). Whenever an event happens strictly earlier or later than at an integral distance from some other event, the difference between the corresponding clocks grows. Counters in ERA record how many times did the differences grow along the current run. The bigger the counter values are, the finer sampling is needed to accommodate all increases along a corresponding run in sampled semantics. We have established and proved this correspondence formally.

In the second step, we have presented a decision procedure which determines whether there is a bound on the counters of an ERA such that all words accepted without any bound are accepted also when the counters are bounded. By this, we obtain a decision procedure for the question whether a given timed automaton can be sampled without losing any qualitative behaviors. This result also extends the previous results on limitedness of counter automata.

In spite of this positive outcome, our results show a high degree of complexity present in dense time behaviors enforced by strict inequalities. Therefore, when we require from our model that it can be turned into a sampled implementation, we have to consider usage of strict inequalities with a great care. It is questionable whether the modeling advantages of strict inequalities outweigh the costs of sampling analysis.

To increase understanding of the relation between dense time models and sampled implementations, we propose to study dense time formalisms which allows us to disregard a concrete sampling rate at modeling time (i.e., some systems may need finer sampling rate than others) and always allows for sampling. Are there such formalisms? If yes, what are their properties?

Counter automata earned their importance by being a simple yet powerful extension of finite automata. Further investigation of R-automata properties would develop tools for understanding the nature of cyclic behaviors in finite state systems. For instance, decidability of limitedness for alternating R-automata is an open problem. An alternating R-automaton might contain states for which computations continuing to all successors have to use the counters in a bounded way.

Communicating Timed Automata.

Channel systems were extensively investigated earlier in an asynchronous setting, where individual machines could infer timing information about other machines only from the received messages. In this thesis, we have shown that time synchronization, i.e., common knowledge of global time, makes channel systems more expressive and by this also more difficult to verify. Interestingly, dense time does not contribute to the qualitative (untimed) expressive power as all results can be obtained even with synchronization on integer timepoints (discrete time) only.

Classical results for channel machines show that giving a finite feedback from the receiver to the sender makes this model Turing complete. Finiteness of the feedback means that the receiver can repeatedly send binary information, which is received instantaneously. We have shown that the amount of information exchanged between the automata implicitly by running in a synchronous setting is smaller than what can be achieved by a finite feedback. This allowed us to develop a procedure which answers several verification questions such as reachability and boundedness for channel systems with the simplest non-trivial topology – one sender and one receiver. On the other hand, unidirectional communication between three timed automata increases the expressive power to recursively enumerable languages.

The analysis of hard real-time control systems where several computational processes communicate through buffers poses a challenge for real-time verification. A plausible solution requires development of the theoretical background as well as implementation of scalable methods for analysis of synchronized channel systems. The initial attempt to tackle this problem presented here is only a first step on this way and leaves many questions open.

The fact that the negative results from asynchronous setting transfer to timed setting means that there are only two other topologies left for which the expressiveness is not known. The first one contains three automata, two senders and one receiver listening to both of them. The other one consists of one sender and two receivers, where the sender transmits to both listeners. More decidability results may be obtained if we restrict ourselves to messages of one type and study cyclic topologies. Another direction is to determine the exact role of the urgent reading for the expressive power of communicating timed automata.

5. References

- [1] Y. Abdeddaïm and O. Maler. Job-shop scheduling using timed automata. In *CAV'01: Proceedings of the 13th International Conference on Computer Aided Verification*, volume 2102 of *Lecture Notes in Computer Science*, pages 478–492. Springer-Verlag, 2001.
- [2] Y. Abdeddaïm and O. Maler. Preemptive job-shop scheduling using stopwatch automata. In *TACAS'02: Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 2280 of *Lecture Notes in Computer Science*, pages 113–126. Springer-Verlag, 2002.
- [3] K. Altisen, G. Gössler, A. Pnueli, J. Sifakis, S. Tripakis, and S. Yovine. A framework for scheduler synthesis. In *RTSS'99: Proceedings of the 20th IEEE Real-Time Systems Symposium*, pages 154–163. IEEE Computer Society Press, 1999.
- [4] K. Altisen and S. Tripakis. Implementation of timed automata: An issue of semantics or modeling? In *FORMATS'05: Proceedings of the 3rd International Conference on Formal Modelling and Analysis of Timed Systems*, volume 3829 of *Lecture Notes in Computer Science*, pages 273–288. Springer-Verlag, 2005.
- [5] R. Alur, C. Courcoubetis, and D. Dill. Model-checking in dense real-time. *Journal of Information and Computation*, 104(1):2–34, 1993.
- [6] R. Alur and D. L. Dill. Automata for modeling real-time systems. In *ICALP'90: Proceedings of the 17th International Colloquium on Automata, Language and Programming*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. Springer-Verlag, 1990.
- [7] R. Alur and D. L. Dill. A theory of timed automata. *Journal of Theoretical Computer Science*, 126(2):183–235, 1994.
- [8] R. Alur, T. Feder, and T. A. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM*, 43(1):116–146, 1996.
- [9] R. Alur and T. A. Henzinger. A really temporal logic. *Journal of the ACM*, 41(1):181–204, 1994.
- [10] R. Alur and P. Madhusudan. Decision problems for timed automata: A survey. In *SFM-RT'04: Proceedings of the 4th International School on Formal Methods for the Design of Computer, Communication and Software Systems: Real Time*, volume 3185 of *Lecture Notes in Computer Science*, pages 1–24. Springer-Verlag, 2004.

- [11] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi. TIMES: a tool for schedulability analysis and code generation of real-time systems. In *FORMATS'03: Proceedings of the 1st International Workshop on Formal Modelling and Analysis of Timed Systems*, number 2791 in *Lecture Notes in Computer Science*, pages 60–72. Springer-Verlag, 2004.
- [12] T. Amnell, E. Fersman, P. Pettersson, W. Yi, and H. Sun. Code synthesis for timed automata. *Nordic Journal of Computing*, 9(4):269–300, 2002.
- [13] E. Asarin, O. Maler, and A. Pnueli. On discretization of delays in timed automata and digital circuits. In *CONCUR'98: Proceedings of the 9th International Conference on Concurrency Theory*, volume 1466 of *Lecture Notes in Computer Science*, pages 470–484. Springer-Verlag, 1998.
- [14] B. Berthomieu and M. Diaz. Modeling and verification of timed dependent systems using timed petri nets. *IEEE Transactions on Software Engineering*, 17(3):259–273, 1991.
- [15] M. Bojańczyk and T. Colcombet. Bounds in omega-regularity. In *LICS'06: Proceedings of the 21st Annual IEEE Symposium on Logic in Computer Science*, pages 285–296. IEEE Computer Society Press, 2006.
- [16] D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2):323–342, 1983.
- [17] G. C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Kluwer Academic Publishers, 1997.
- [18] G. Cécé and A. Finkel. Verification of programs with half-duplex communication. *Journal of Information and Computation*, 202(2):166–190, 2005.
- [19] J. H. Chang, O. H. Ibarra, and A. Vergis. On the power of one-way communication. *Journal of the ACM*, 35(3):697–726, 1988.
- [20] Z. Chaochen. Duration calculus, a logical approach to real-time systems. In *AMAST'98: Proceedings of the 7th International Conference on Algebraic Methodology and Software Technology*, volume 1548 of *Lecture Notes in Computer Science*, pages 1–7. Springer-Verlag, 1999.
- [21] Z. Chaochen, C. A. R. Hoare, and A. P. Ravn. A calculus of duration. *Information Processing Letters*, 40(5):269–276, 1991.
- [22] T. Colcombet and C. Löding. The nesting-depth of disjunctive μ -calculus for tree languages and the limitedness problem. In *CSL'08: Proceedings of the 17th EACSL Annual Conference on Computer Science Logic*, volume 5213 of *Lecture Notes in Computer Science*, pages 416–430. Springer-Verlag, 2008.
- [23] L. de Alfaro, M. Faella, T. A. Henzinger, R. Majumdar, and M. Stoelinga. The element of surprise in timed games. In *CONCUR'03: Proceedings of the 14th International Conference on Concurrency Theory*, volume 2761 of *Lecture Notes in Computer Science*, pages 144–158. Springer-Verlag, 2003.

- [24] D. D'Souza. A logical characterisation of event recording automata. In *FTRFTT'00: Proceedings of the 6th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 1926 of *Lecture Notes in Computer Science*, pages 240–251. Springer-Verlag, 2000.
- [25] C. Ericsson, A. Wall, and W. Yi. Timed automata as task models for event-driven systems. In *RTCSA'99: Proceedings of the 6th International Workshop on Real-Time Computing and Applications Symposium*. IEEE Computer Society Press, 1999.
- [26] T. A. Henzinger F. Cassez and J.-F. Raskin. A comparison of control problems for timed and hybrid systems. In *HSCC'02: Proceedings of the 5th International Workshop on Hybrid Systems: Computation and Control*, volume 2289 of *Lecture Notes in Computer Science*, pages 134–148. Springer-Verlag, 2002.
- [27] E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi. Schedulability analysis using two clocks. In *TACAS'03: Proceedings of the 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, number 2619 in *Lecture Notes in Computer Science*, pages 224–239. Springer-Verlag, 2003.
- [28] E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi. Schedulability analysis of fixed-priority systems using timed automata. *Journal of Theoretical Computer Science*, 354:301–317, 2006.
- [29] E. Fersman and W. Yi. A generic approach to schedulability analysis of real-time tasks. *Nordic Journal of Computing*, 11(2):129–147, 2004.
- [30] A. Finkel and P. McKenzie. Verifying identical communicating processes is undecidable. *Journal of Theoretical Computer Science*, 174(1-2):217–230, 1997.
- [31] A. Göllü, A. Puri, and P. Varaiya. Discretization of timed automata. In *CDC'94: Proceedings of the 33rd IEEE Conference on Decision and Control*, pages 957–958, 1994.
- [32] O. Grinchtein. *Learning of Timed Systems*. PhD thesis, Uppsala University, Department of Information Technology, 2008.
- [33] K. Hashiguchi. Limitedness theorem on finite automata with distance functions. *Journal of Computer and System Sciences*, 24(2):233–244, 1982.
- [34] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57(1):94–124, 1998.
- [35] T. A. Henzinger, Z. Manna, and A. Pnueli. Temporal proof methodologies for real-time systems. In *POPL'91: Proceedings of the 18th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 353–366, New York, NY, USA, 1991. ACM Press.
- [36] T. A. Henzinger, Z. Manna, and A. Pnueli. What good are digital clocks? In *ICALP'92: Proceedings of the 19th International Colloquium on Algorithms Languages and Programming*, volume 623 of *Lecture Notes in Computer Science*, pages 545–558. Springer-Verlag, 1992.

- [37] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Journal of Information and Computation*, 111(2):193–244, 1994.
- [38] F. Jahanian and A. K. Mok. Safety analysis of timing properties in real-time systems. *IEEE Transactions on Software Engineering*, 12(9):890–904, 1986.
- [39] F. Jahanian and A. K. Mok. Modechart: A specification language for real-time systems. *IEEE Transactions on Software Engineering*, 20(12):933–947, 1994.
- [40] D. Kirsten. Distance desert automata and the star height problem. *Informatique Theorique et Applications*, 39(3):455–509, 2005.
- [41] R. Koymans. Specifying real-time properties with metric temporal logic. *Real Time Systems*, 2(4):255–299, 1990.
- [42] P. Krčál, L. Mokrushin, P. S. Thiagarajan, and W. Yi. Timed vs. time triggered automata. In *CONCUR'04: Proceedings of the 15th International Conference on Concurrency Theory*, volume 3170 of *Lecture Notes in Computer Science*, pages 340–354. Springer-Verlag, 2004.
- [43] P. Krčál and R. Pelánek. On sampled semantics of timed systems. In *FSTTCS'05: Proceedings of the 14th International Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 3821 of *Lecture Notes in Computer Science*, pages 310–321. Springer-Verlag, 2005.
- [44] P. Krčál and W. Yi. Decidable and undecidable problems in schedulability analysis using timed automata. In *TACAS'04: Proceedings of the 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 2988 of *Lecture Notes in Computer Science*, pages 236–250. Springer-Verlag, 2004.
- [45] H.-H. Kwak, I. Lee, A. Philippou, J.-Y. Choi, and O. Sokolsky. Symbolic schedulability analysis of real-time systems. In *RTSS'98: Proceedings of the 19th IEEE Real-Time Systems Symposium*, pages 409–418. IEEE Computer Society Press, 1998.
- [46] K. G. Larsen, P. Petterson, and W. Yi. UPPAAL in a nutshell. *Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.
- [47] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.
- [48] J. W. S. Liu. *Real-Time Systems*. Prentice-Hall, 2000.
- [49] N. Lynch and H. Attiya. Using mappings to prove timing properties. In *PODC'90: Proceedings of the 9th annual ACM symposium on Principles of distributed computing*, pages 265–280. ACM Press, 1990.
- [50] J. McManis and P. Varaiya. Suspension automata: A decidable class of hybrid automata. In *CAV'94: Proceedings of the 6th International Conference on Computer Aided Verification*, volume 818 of *Lecture Notes in Computer Science*, pages 105–117. Springer-Verlag, 1994.

- [51] P. Merlin and D. Farber. Recoverability of communication protocols—implications of a theoretical study. *IEEE Transactions on Communications*, 24(9):1036–1043, 1976.
- [52] A. K. Mok, D.-C. Tsou, and R. C. M. de Rooij. The MSP.RTL real-time scheduler synthesis tool. In *RTSS'96: Proceedings of the 17th IEEE Real-Time Systems Symposium*, pages 118–128. IEEE Computer Society Press, 1996.
- [53] X. Nicollin and J. Sifakis. The algebra of timed processes, ATP: Theory and application. *Journal of Information and Computation*, 114(1):131–178, 1994.
- [54] J. Ouaknine and J. Worrell. Universality and language inclusion for open and closed timed automata. In *HSCC'03: Proceedings of the 6th International Workshop on Hybrid Systems: Computation and Control*, volume 2623 of *Lecture Notes in Computer Science*, pages 375–388. Springer-Verlag, 2003.
- [55] J. K. Pachl. Reachability problems for communicating finite state machines. Technical Report CS-82-12, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, May 1982.
- [56] G. M. Reed and A. W. Roscoe. A timed model for communicating sequential processes. *Journal of Theoretical Computer Science*, 58(1-3):249–261, 1988.
- [57] I. Simon. Factorization forests of finite height. *Journal of Theoretical Computer Science*, 72(1):65–94, 1990.
- [58] I. Simon. On semigroups of matrices over the tropical semiring. *Informatique Theorique et Applications*, 28(3-4):277–294, 1994.
- [59] T. Wilke. Specifying timed state sequences in powerful decidable logics and timed automata. In *FTRFT'94: Proceedings of the 3rd on Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 863 of *Lecture Notes in Computer Science*, pages 694–715. Springer-Verlag, 1994.
- [60] M. De Wulf, L. Doyen, and J.-F. Raskin. Almost ASAP semantics: From timed models to timed implementations. In *HSCC'04 : Proceedings of the 7th International Workshop on Hybrid Systems: Computation and Control*, volume 2993 of *Lecture Notes in Computer Science*, pages 296–310. Springer-Verlag, 2004.
- [61] W. Yi. CCS + time = an interleaving model for real time systems. In *ICALP'91: Proceedings of the 18th International Colloquium on Automata, Language and Programming*, volume 510 of *Lecture Notes in Computer Science*, pages 217–228. Springer-Verlag, 1991.
- [62] S. Yovine. Kronos: a verification tool for real-time systems. *Journal on Software Tools for Technology Transfer*, 1(1-2):123–133, 1997.

Acta Universitatis Upsaliensis

*Digital Comprehensive Summaries of Uppsala Dissertations
from the Faculty of Science and Technology 633*

Editor: The Dean of the Faculty of Science and Technology

A doctoral dissertation from the Faculty of Science and Technology, Uppsala University, is usually a summary of a number of papers. A few copies of the complete dissertation are kept at major Swedish research libraries, while the summary alone is distributed internationally through the series Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology. (Prior to January, 2005, the series was published under the title “Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology”.)

Distribution: publications.uu.se
urn:nbn:se:uu:diva-100549



ACTA
UNIVERSITATIS
UPSALIENSIS
UPPSALA
2009

Paper I



Task Automata: Schedulability, Decidability and Undecidability

Elena Fersman¹, Pavel Krcal, Paul Pettersson², and Wang Yi³

Department of Information Technology
Uppsala University, Sweden
Email: {pavelk, paupet, yi}@it.uu.se

Abstract. We present a model, *task automata*, for real time systems with non-uniformly recurring computation tasks. It is an extended version of timed automata with asynchronous processes that are computation tasks generated (or triggered) by timed events. Compared with classical task models for real time systems, task automata may be used to describe tasks (1) that are generated non-deterministically according to timing constraints in timed automata, (2) that may have interval execution times representing the best case and the worst case execution times, and (3) whose completion times may influence the releases of task instances. We generalize the classical notion of schedulability to task automata. A task automaton is schedulable if there exists a scheduling strategy such that all possible sequences of events generated by the automaton are schedulable in the sense that all associated tasks can be computed within their deadlines. Our first technical result is that the schedulability for a given scheduling strategy can be checked algorithmically for the class of task automata when the best case and the worst case execution times of tasks are equal. The proof is based on a decidable class of suspension automata: timed automata with bounded subtraction in which clocks may be updated by subtractions within a bounded zone. We shall also study the borderline between decidable and undecidable cases. Our second technical result shows that the schedulability checking problem will be undecidable if the following three conditions hold: (1) the execution times of tasks are intervals, (2) the precise finishing time of a task instance may influence new task releases, and (3) a task is allowed to preempt another running task.

¹Current address: Ericsson AB, Torshamnsgatan 23, SE-164 80 Stockholm, Sweden. Email: elena.fersman@ericsson.com.

²Current address: Department of Computer Science and Electronics, Mälardalen University, Sweden. Email: Paul.Pettersson@mdh.se.

³Corresponding author: Wang Yi, Box 337, 751 05, Uppsala, Sweden. Email: yi@it.uu.se. Tel: +46 18 4713110.

1 Introduction

One of the most important issues in developing real time systems is *schedulability analysis* prior to implementation. In the area of real time scheduling, there are well-studied methods [1], e.g., rate monotonic scheduling, that are widely applied in the analysis of periodic tasks with deterministic behaviours. For *non-periodic* tasks with non-deterministic behaviours, there are no satisfactory solutions. There are approximative methods with pessimistic analysis, e.g., using periodic tasks to model sporadic tasks when control structures of tasks are not considered. The advantage of automata-theoretic approaches, e.g., using timed automata in modeling systems is that one may specify general timing constraints on events and model other behavioural aspects such as concurrency and synchronization. However, it is not clear how timed automata can be used for schedulability analysis because there is no support for specifying resource requirements and hard time constraints on computations, e.g., deadlines.

Following the work of [2], we study task automata, that are timed automata extended with real time tasks triggered by events. A task is an executable program characterized by its best case and worst case execution time, deadline, and possibly other parameters such as priorities for scheduling. The main idea is to associate each location of a timed automaton with a task (or a set of tasks in the general case). Intuitively a discrete transition leading to a location in the automaton denotes an event triggering an instance of the annotated task and the guard (clock constraints) on the transition specifies the possible arrival times of the event. Semantically, an automaton may perform two types of transitions. Delay transitions correspond to the execution of a running task (with the highest priority) and idling for the other tasks. Discrete transitions correspond to the arrival of new task instances. Whenever a task is triggered, it will be put into the scheduling queue for execution (i.e., the ready queue in operating systems). We assume that the tasks will be executed according to a given scheduling strategy, e.g., FPS (fixed priority scheduling) or EDF (earliest deadline first).

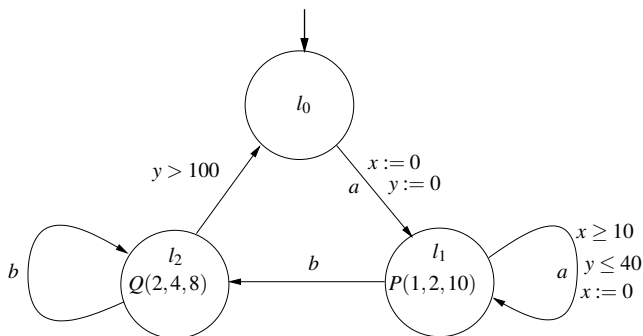


Figure 1.1: A task automaton.

For example, consider the automaton shown in Figure I.1. It has three locations l_0, l_1, l_2 , and two tasks P and Q (triggered by a and b) with interval computation times $[1, 2]$ and $[2, 4]$ (the best case and the worst case execution times), and relative deadlines 10 and 8, respectively. The automaton models a system starting in l_0 that may move to l_1 by event a at any time. This triggers the task P . In l_1 , as long as the constraints $x \geq 10$ and $y \leq 40$ hold, when an event a occurs an instance of task P will be created and put into the scheduling queue. However, it cannot create more than 5 instances of P in l_1 , because the constraint $y \leq 40$ will be violated after 40 time units. In fact, every instance will be computed before the next instance arrives and the scheduling queue may contain at most one task instance. Therefore, no task instance of P will miss its deadline. The system is also able to accept b , switch from l_1 to l_2 , and trigger Q . Because there are no constraints labeled on the b -transition in l_2 , it may accept any number of b 's and create any number of Q 's in zero time. However, after more than two copies of Q , the queue will be non-schedulable i.e. a deadline may be violated. This means that the system is non-schedulable. Thus, zero behaviours will correspond to non-schedulability, which is a natural property of the model.

We shall formalize the notion of schedulability in terms of reachable states. A state of a task automaton will be a triple (l, u, q) consisting of a location l , a clock valuation u , and a task queue q . The task queue contains pairs of remaining computation times and relative deadlines for all released tasks. A scheduling strategy is a function on queues, which inserts a new task instance into the task queue according to the task parameters such as fixed priorities, remaining computation times, and (or) deadlines. Formally, we assume that a scheduling strategy can be encoded as timed automata. We shall see that the existing scheduling strategies (preemptive or non-preemptive) in the literature such as EDF or FPS satisfy this condition. An automaton is schedulable if there exists a scheduling strategy with which all tasks in q can be computed within their deadlines for all reachable states (l, u, q) of the automaton.

In [2], it is shown that under the assumption that the tasks are non-preemptive, the schedulability checking problem for a given (non-preemptive) scheduling strategy can be transformed to a reachability problem for ordinary timed automata and thus it is decidable. For *preemptive scheduling* strategies, it has been suspected that the schedulability checking problem is undecidable because in preemptive scheduling we must use stopwatches to accumulate computation times for tasks. In this paper, we show that to model the scheduling problems, the expressive power of stopwatch automata is not needed.

Our main technical result is that the schedulability checking problem related to a preemptive scheduling strategy is decidable for a large class of task automata. We show that the problem is decidable if the best case and the worst case computation times of tasks are equal. The crucial observation in the proof is that the schedulability checking problem can be translated to a reachability problem for a decidable class of suspension automata [3] – timed automata with bounded subtraction where clocks may be updated with subtraction only

in a bounded zone. We also show that the case with variable execution times can be reduced to the previous one for FPS and EDF if the finishing times of tasks do not influence release times of new tasks.

In particular, the schedulability problem related to EDF can be checked for these classes of automata. EDF is optimal in the sense that if it cannot schedule a task queue, no other scheduling strategy can. Therefore, the general schedulability checking problem for a task automaton (whether there is a strategy which can schedule the automaton) can be checked if the best case and the worst case computation times of tasks are equal or if the finishing times of tasks do not influence release times of new tasks.

We shall also study the borderline between decidable and undecidable cases. It is shown that the schedulability problem for preemptive scheduling strategies is undecidable if task execution times may vary within an interval representing the best and the worst case execution times. This is a surprisingly negative result that such a subtle difference can turn a decidable problem to an undecidable one. More precisely, the schedulability problem for many scheduling strategies is undecidable if these three conditions hold: (1) the execution times of tasks are intervals, (2) the precise finishing time of a task may influence the new task releases and (3) a task is allowed to preempt another running task.

The rest of this paper is organized as follows: Section 2 describes the notions of task and scheduling strategy, and the syntax and semantics of task automata. Section 3 describes scheduling problems related to task automata with a summary on decidability and undecidability results. Section 4 is devoted to proofs for the decidable cases. Section 5 presents a proof for the undecidable case. Section 6 concludes the paper with summarized results and future work, and a brief summary on and comparison with related work.

2 Timed Automata with Tasks

We extend timed automata with asynchronous processes, i.e., tasks triggered by events and computed asynchronously. The main idea is to associate each location of a timed automaton with an abstraction of an executable program called a *task type* or simply a task.

2.1 Tasks and Scheduling Strategy

Assume a set of *task types* \mathcal{P} ranged over by P, Q, R , etc. A task type or simply a task may have task parameters such as fixed priority, computation time (the best and the worst case), deadline, and resource requirements, e.g., on memory consumption. For simplicity, we do not consider resource requirements in this paper. Assume that B, W, D are three natural numbers such that $B \leq W \leq D$ and $0 < W$. A task type is a tuple (P, B, W, D) , written $P(B, W, D)$, where P is the task name, B is the *best case execution time*, W is the *worst case execution time* and D is the *relative deadline*. Note that D is a relative deadline meaning

that whenever an instance of P is released, it should be computed within D time units. We sometimes say that the computation time of P is in $[B, W]$. When a set of tasks is scheduled according to fixed priorities then we assume that each task type has assigned a priority.

A task may have several instances that are different copies of the same program. A task instance is a tuple (P, b, w, d) , written $P(b, w, d)$, where P is a task name, $b \in \mathbb{R}$ is a best case remaining computation time, $w \in \mathbb{R}$ is a worst case remaining computation time, and $d \in \mathbb{R}$ is a relative deadline. We shall use p_i to denote a task instance and, without confusion, p_i 's task type will be understood as (P_i, B_i, W_i, D_i) . Note that different task instances may be of the same task type with the same task parameters. A task queue is a list of task instances denoted $[P_1(b_1, w_1, d_1), \dots, P_n(b_n, w_n, d_n)]$. The discrete part of a queue is a list containing only the task names. A set of all task queues containing instances of the task types from \mathcal{P} is denoted $Q_{\mathcal{P}}$.

We shall study scheduling problems for single-processor systems. Thus we assume that a task queue is a sorted list whose head element is the task instance running on the processor, and the other ones are waiting. A scheduling strategy, e.g., FPS (fixed priority scheduling), SJF (shortest job first), or EDF (earliest deadline first), is a function which inserts released tasks into the task queue.

More precisely, a *scheduling strategy* (or scheduling function) $\text{Sch} : \mathcal{P} \times Q_{\mathcal{P}} \mapsto Q_{\mathcal{P}}$ is a function which given a task instance and a task queue returns a task queue with the task instance inserted and the order of the other task instances preserved. For example, $\text{EDF}(P(1, 3, 10), [Q(3, 4, 5.3), R(0, 2, 19)]) = [Q(3, 4, 5.3), P(1, 3, 10), R(0, 2, 19)]$. A scheduling strategy has to satisfy the following condition. The decision on where the new task instance is inserted in the queue can be made only by comparing the task parameters of the new task instance with each of the existing instances in the queue and by considering the discrete part of the queue. The task parameters are either the remaining best and worst case computation times, or the remaining relative deadlines. We formalize the concept of a scheduling strategy in terms of timed automata in Definition 7.

Scheduling strategies may be preemptive or non-preemptive:

1. A non-preemptive strategy will never insert the new task as the first element of the queue.
2. A preemptive strategy may insert the new task in the first position if its task type is different from the current running task and all suspended (pre-empted) tasks in the queue.

To talk about computation and resource consumption, we shall use a function $\text{Run} : Q_{\mathcal{P}} \times \mathbb{R}_{\geq 0} \mapsto Q_{\mathcal{P}}$ which given a real number t and a task queue q returns the task queue after t time units of execution on a processor. The result of $\text{Run}(q, t)$ for $t \leq w_1$ and $q = [P(b_1, w_1, d_1), Q(b_2, w_2, d_2), \dots, R(b_n, w_n, d_n)]$ is defined as $q' = [P(b_1 - t, w_1 - t, d_1 - t), Q(b_2, w_2, d_2 - t), \dots, R(b_n, w_n, d_n - t)]$. For example, let $q = [Q(2, 3, 5), P(4, 7, 10)]$. Then $\text{Run}(q, 3) = [Q(-1, 0, 2),$

$P(4, 7, 7]$ in which the first task has been executed for 3 time units (and it will be removed from the queue).

A task instance $P(b, w, d)$ in the queue may finish when $b \leq 0$ and $w \geq 0$, and it must finish when $w = 0$. Finished tasks are removed from the queue. This is reflected in the definition of semantics of task automata, Definition 2.

2.2 Task Automata

As in timed automata, assume a finite alphabet \mathbb{Act} ranged over by a, b, \dots and a finite set of real-valued clocks \mathcal{C} ranged over by x_1, x_2, \dots . We use $\mathcal{B}(\mathcal{C})$ ranged over by g to denote the set of conjunctive formulas of atomic constraints in the form: $x_i \bowtie C$ or $x_i - x_j \bowtie C$ where $x_i, x_j \in \mathcal{C}$ are clocks, $\bowtie \in \{\leq, <, \geq, >\}$, and C is a natural number. The elements of $\mathcal{B}(\mathcal{C})$ are called *clock constraints*.

Assume a distinguished clock x_{done} which is reset every time a task finishes its computation and is removed from the task queue. This clock can be used to model data dependencies or precedence relations between tasks. One may introduce such a clock or a boolean for every task type without changing the technical results of this paper. It can be easily seen from the decidability proofs that more such clocks or boolean variables can be accommodated and that it is enough to have one clock for the undecidability result. Therefore, for simplicity of presentation, we use only x_{done} .

Definition 1 A task automaton over actions \mathbb{Act} , clocks \mathcal{C} , and task types \mathcal{P} is a tuple $\langle N, l_0, E, I, M, x_{done} \rangle$ where

- N is a finite set of locations ranged over by l, m, n ,
- $l_0 \in N$ is the initial location,
- $E \subseteq N \times \mathcal{B}(\mathcal{C}) \times \mathbb{Act} \times 2^{\mathcal{C}} \times N$ is the set of edges,
- $I : N \mapsto \mathcal{B}(\mathcal{C})$ is a function assigning each location with a clock constraint (a location invariant),
- $M : N \mapsto \mathcal{P}$ is a partial function assigning locations with task types,⁴ and
- $x_{done} \in \mathcal{C}$ is the clock which is reset whenever a task finishes.

When $\langle l, g, a, r, l' \rangle \in E$, we write $l \xrightarrow{g a r} l'$.

A task automaton is said to have no task feedback if none of its guards or invariants contains the clock x_{done} . Further, a task automaton has fixed computation times of tasks if $B = W$ for all task types $P(B, W, D)$.

Note that the distinguished clock x_{done} may be used in the guards or invariants of a task automaton, which means that the finishing times of tasks may influence the behaviour of the task automaton. However, the finishing time of a task does not have any influence on the behaviour of a task automaton when it has no task feedback.

⁴ Note that M is a partial function meaning that some of the locations may have no tasks.

2.3 Operational Semantics

Similarly to timed automata, a task automaton may perform two types of transitions. Delay transitions correspond to the execution of the running task (e.g., a task with the highest priority or with the earliest deadline) and idling for the other tasks waiting to run. These transitions are split into three subtypes according to the queue status (empty, non-empty) and the fact, whether a task finishes. Discrete transitions correspond to the arrivals of new task instances.

We represent the values of clocks as functions (i.e., clock assignments) from \mathcal{C} to the non-negative reals $\mathbb{R}_{\geq 0}$. We denote by \mathcal{V} the set of clock assignments for \mathcal{C} . Naturally, a semantic state of an automaton is a triple (l, u, q) where l is the current location, $u \in \mathcal{V}$ denotes the current values of clocks, and q is the current task queue. By u_0 we denote a clock assignment such that $u_0(x) = 0$ for all clocks x .

We use $u \models g$ to denote that the clock assignment u satisfies the constraint g . For $t \in \mathbb{R}_{\geq 0}$, we use $u + t$ to denote the clock assignment which maps each clock x to the value $u(x) + t$, and $u[r]$ for $r \subseteq \mathcal{C}$, to denote the clock assignment which maps each clock in r to 0 and agrees with u for the other clocks (i.e., $\mathcal{C} \setminus r$). Now we are ready to present the operational semantics for task automata as labeled transition systems (LTS).

Definition 2 *Given a scheduling strategy Sch , the semantics of an automaton $A = \langle N, l_0, E, I, M, x_{done} \rangle$ is a labeled transition system $\llbracket A_{\text{Sch}} \rrbracket$ with an initial state (l_0, u_0, \square) and transitions defined by the following rules:*

- $(l, u, q) \xrightarrow{a}_{\text{Sch}} (l', u[r], \text{Sch}(M(l'), q))$ if $l \xrightarrow{g^{\text{ar}}} l'$, $u \models g$, and $u[r] \models I(l')$,
 - $(l, u, \square) \xrightarrow{t}_{\text{Sch}} (l, u + t, \square)$ if $t \in \mathbb{R}_{\geq 0}$ and $(u + t) \models I(l)$,
 - $(l, u, P(b, w, d) :: q) \xrightarrow{t}_{\text{Sch}} (l, u + t, \text{Run}(P(b, w, d) :: q, t))$ if $t \in \mathbb{R}_{\geq 0}$, $t \leq w$ and $(u + t) \models I(l)$, and
 - $(l, u, P(b, w, d) :: q) \xrightarrow{\text{fin}}_{\text{Sch}} (l, u[x_{done}], q)$ if $b \leq 0 \leq w$ and $u[x_{done}] \models I(l)$,
- where $P(b, w, d) :: q$ denotes the queue with the task instance $P(b, w, d)$ inserted into q (at the first position), \square denotes the empty queue, and $\text{fin} \notin \mathbb{A}$ is a distinct action name.

Note that the transition rules are parameterized by Sch (scheduling strategy). Whenever it is understood from the context, we shall omit Sch from the transition relation. We have the same notion of reachability as for timed automata.

Definition 3 *We shall write $(l, u, q) \longrightarrow_{\text{Sch}} (l', u', q')$ if either $(l, u, q) \xrightarrow{a}_{\text{Sch}} (l', u', q')$ for an action a , or $(l, u, q) \xrightarrow{t}_{\text{Sch}} (l', u', q')$ for a delay t , or $(l, u, q) \xrightarrow{\text{fin}}_{\text{Sch}} (l', u', q')$. For a task automaton with initial state (l_0, u_0, \square) and a scheduling strategy Sch , (l, u, q) is reachable iff $(l_0, u_0, \square) \longrightarrow_{\text{Sch}}^* (l, u, q)$.*

Clearly task automata are at least as expressive as timed automata. In fact, $\langle N, l_0, E, I \rangle$ is an ordinary timed automaton. Intuitively, a discrete transition of the automaton denotes an event triggering a task annotated in the target

location, and the guard on the edge specifies all the possible arrival times of the event (or the annotated task). Whenever a task is triggered, it will be put into the scheduling (or task) queue for execution (corresponding to the ready queue in operating systems). In general, the task queue is unbounded though the constraints of a given automaton may restrict the possibility of reaching states with infinitely many different task queues. For example, we may model time-triggered periodic tasks as a simple automaton as shown in Figure I.2(a) where P is a periodic task with computation time in $[1, 2]$, deadline 8 and period 20. More generally, it may model systems containing both periodic and sporadic tasks as shown in Figure I.2(b) which is a system consisting of 4 tasks as annotation on locations, where P and Q are triggered by time every 20 and 40 time units respectively (specified by the constraints: $x = 20$ and $x = 40$), and R and S are sporadic or event driven by event a and b respectively.

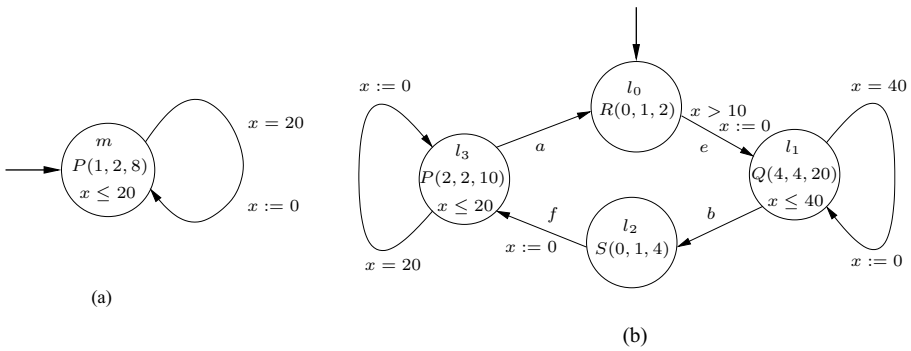


Figure I.2: Modeling Periodic and Sporadic Tasks.

To handle concurrency and synchronization, a parallel composition of task automata may be defined as a product automaton in the same way as for ordinary timed automata (e.g., see [4]). The only difference is that a product location might be assigned more than one task. In such a case, auxiliary locations are introduced so that each of them has at most one task assigned, the tasks are released at the same timepoint, and all interleavings (all orders in which the tasks are released) are allowed. Note that the parallel composition here is only an operator to construct models of systems based on their components. It has nothing to do with multi-processor scheduling.

As standard timed automata, the underlying timed automata of task automata may have time converging behaviours known as *timestops*, that are sequences of transitions where time does not diverge. Consider, for example, a location with no outgoing transitions and an invariant $x < 10$ for a clock x . Clearly, the automaton cannot delay in this location for longer than 10 time units. This would imply that some tasks in the queue never finish their computation if it requires more than 10 time units. Also, as time is not progressing, no deadline will be violated either. In order to simplify technical details in the proofs we prohibit timestops caused by timed automata invariants. However,

this is not fundamental for our results and it is not difficult to check that it is also possible to handle a situation when the time does not progress. We refer to the standard semantics ([5]) of timed automata in the next definition.

Definition 4 A task automaton $A = \langle N, l_0, E, I, M, x_{done} \rangle$ has no timesteps if for the underlying timed automaton $A' = \langle N, l_0, E, I \rangle$, for any state (l, u) of A' , and all $L \in \mathbb{R}_{\geq 0}$ there exists a path $(l, u) \xrightarrow{*} (l', u')$ in the semantics LTS of A' such that the sum of the time delays on the path from (l, u) to (l', u') is equal to L .

From now on, we assume that task automata have no timesteps which implies that each task will be eventually computed in such automata.

To demonstrate the semantics of task automata, consider the automaton in Figure I.2(b). Assume that preemptive earliest deadline first (EDF) strategy is used to schedule the task queue. For example, the automaton may stay in l_1 or l_2 where instances of the periodic tasks P and Q may be released and computed. Or it may loop from l_0 back to l_0 through l_1 , l_2 , and l_3 . Though the queue is growing during these transitions, the generation of new task instances will be slowed down by the constraint $x > 10$ labeled on the edge from l_0 to l_1 and the queue will be reduced by delay transitions, e.g.,

$$(l_0, [x = 0, x_{done} = 0], [S(-0.3, 0.7, 3.7), R(0, 1, 2), P(2, 2, 8), Q(3.5, 3.5, 9.2)]) \\ \xrightarrow{+} (l_0, [x = 10, x_{done} = 3.1], []).$$

A question of interest is whether it can perform a sequence of transitions leading to a state where a deadline has been missed.

3 Schedulability Analysis

In this section we study verification problems related to the model presented in the previous section. One of the most interesting properties of task automata related to the task queue is schedulability.

3.1 Schedulability of Task Automata

As all deadlines in task automata are hard, we define schedulability for a given scheduling strategy as impossibility of reaching a state where some deadline is missed. We use q_{err} to denote queues containing a task instance $P(b, w, d)$ with $d < 0$.

Definition 5 (Schedulability) A task automaton A with initial state $(l_0, u_0, [])$ is non-schedulable with Sch if $(l_0, u_0, []) \xrightarrow{*}_{\text{Sch}} (l, u, q_{err})$ for some l and u . Otherwise, we say that A is schedulable with Sch. More generally, we say that A is schedulable if and only if there exists a scheduling strategy Sch with which A is schedulable.

We also need a notion of non-schedulable queues that are queues which will inevitably lead to a deadline miss with time progress (if all tasks take their worst case computation times).

Definition 6 A queue $q = [P_1(b_1, w_1, d_1) \dots P_n(b_n, w_n, d_n)]$ is non-schedulable if $w_1 + \dots + w_i > d_i$ for some $1 \leq i \leq n$.

Even though queues are not bounded in general, an important observation is that all schedulable queues are bounded. First, note that a task instance that has been started cannot be preempted by another instance of the same task type. This means that there is only one instance of each task type in the queue whose computation time can be a real number and it can be arbitrarily small. Thus the number of instances of each task type $P_i \in \mathcal{P}$, in a schedulable queue is bounded by $\lceil D_i/W_i \rceil$ and the size of schedulable queues is bounded by $\sum_{P_i \in \mathcal{P}} \lceil D_i/W_i \rceil$. This is an important property of our model, because it allows us to code schedulability checking problems as reachability problems.

Throughout of the paper, we shall distinguish three situations according to the queue status:

1. A queue is an *error-queue* denoted q_{err} if a deadline is already missed.
2. A queue is *non-schedulable* as defined in Definition 6 if it will inevitably evolve to an error-queue.
3. A queue is *overflowed* if it contains more than $\lceil D_i/W_i \rceil$ instances of P_i for some i .

Note that an overflowed queue is definitely non-schedulable. But a very short (not overflowed) queue can also be non-schedulable. We shall say that a state is *adequate* if its queue is neither an overflowed queue nor an error-queue. However, an adequate state may contain a non-schedulable queue.

Before presenting the results, we formalize the concept of scheduling strategy used in this paper in terms of timed automata. Assume a task type $P(B, W, D)$ and a task queue $[P_1(b_1, w_1, d_1), \dots, P_n(b_n, w_n, d_n)]$. We construct a diagonal-free (i.e. clocks are compared only to constants) timed automaton with clocks $x_{b_1}, x_{w_1}, x_{d_1}, \dots, x_{b_n}, x_{w_n}, x_{d_n}$, $n + 2$, locations $l_0, l_1, l_2, \dots, l_{n+1}$, and $n + 1$ edges from l_0 to l_i for $i \leq n + 1$. We call such an automaton a *decision automaton*.

Definition 7 A scheduling strategy $Sch : \mathcal{P} \times Q_{\mathcal{P}} \mapsto Q_{\mathcal{P}}$ is a function satisfying the following condition. For each task type $P(B, W, D)$ and a task queue $[P_1(b_1, w_1, d_1), \dots, P_n(b_n, w_n, d_n)]$, one can effectively construct a decision automaton such that $Sch(P(B, W, D), [P_1(b_1, w_1, d_1), \dots, P_n(b_n, w_n, d_n)])$ inserts $P(B, W, D)$ into the queue at the k -th position if and only if l_k is the only location reachable from (l_0, u) where $u(x_{b_i}) = b_i, u(x_{w_i}) = w_i, u(x_{d_i}) = d_i$ for all $1 \leq i \leq n$.

Note that to make location l_k reachable, the automaton should be constructed in such a way that the edges from l_0 to l_i for all $i \leq n + 1$ are labeled with guards corresponding to the conditions on the task types and parameters to be checked for the scheduler in making the decision on where the new task

instance, i.e., $P(B, W, D)$, should be inserted into the queue. Note also that the definition corresponds to the informal description on scheduling strategy in Section 2. In particular, the known scheduling strategies such as EDF, SJF, and FPS all satisfy the condition defined.

3.2 Decidability and Undecidability Results

First, we consider the case of non-preemptive scheduling to introduce the problems. We have the following positive result.

Theorem 1 *The problem of checking schedulability relative to a non-preemptive scheduling strategy for task automata is decidable.*

Proof. A detailed proof is given in [2]. We sketch the proof idea here. It is to show that the schedulability question for a task automaton can be translated as a reachability question for a timed automaton.

We transform the underlying timed automaton of a task automaton $A = \langle N, l_0, E, I, M, x_{done} \rangle$ to a modified timed automaton $E(A)$ as follows. We remove all labels, and add a label $release_i$ on all edges leading to a location l such that $M(l) = P_i$. This gives us a possibility to keep the information about which task is released when the automaton enters l .

We code the task queue and operations on the queue related to the given scheduling strategy as a timed automaton (called the scheduler) denoted $E(\text{Sch})$. This automaton remembers the discrete parts of the queues in the locations and it uses clocks to remember the accumulated computation times and the relative deadlines for the released task instances. It is sufficient to encode only the queues from the adequate states. Therefore, there are only finitely many different discrete parts of the queues. The edges of $E(\text{Sch})$ are induced by the timed automata for the scheduling strategy Sch from Definition 7 and by the rules for the task finishing from Definition 2.

The scheduler automaton manipulates the clocks as follows: Whenever an instance of a task type P_i is released by an event $release_i$, a clock x_{ij}^d is reset to 0, for some j such that x_{ij}^d is not used by any other task instance. Whenever a released task instance p_{ij} is started to run, a clock x_{ij}^c is reset to 0. Whenever the value of clock x_{ij}^c of the running task instance is greater than or equal to B_i , the task instance can be removed from the queue and the next task instance can start to run. Note that this value should never be greater than W_i . Whenever the constraints $x_{ij}^d = D_i$ and $x_{ij}^c < W_i$ are met, an error state should be reached. Whenever the scheduling strategy needs to access the remaining best case/worst case computation time or the remaining deadline of a task instance (to compare it with a constant), it can use $B_i - x_{ij}^c$, $W_i - x_{ij}^c$, $D_i - x_{ij}^d$ for the running task instance ($B_i, W_i, D_i - x_{ij}^d$ for a released but not running task instance), respectively.

Finally we construct the product automaton $E(\text{Sch}) \parallel E(A)$ in which both $E(\text{Sch})$ and $E(A)$ can only synchronize on identical actions namely $release_i$'s.

It can be proved that if an error state of the product automaton is reachable, the original task automaton is non-schedulable. ■

For *preemptive scheduling* strategies, it has been conjectured that the schedulability checking problem is undecidable. The reason is that if we use the same ideas as for non-preemptive scheduling to encode a preemptive scheduling strategy, we must use stopwatches (or integrators) to accumulate computation times for suspended tasks. That is, it appears that the computation model behind preemptive scheduling is stopwatch automata for which it is known that the reachability problem is undecidable. However, we have positive results for the following two classes of task automata.

Theorem 2 *The problem of checking schedulability relative to a preemptive scheduling strategy is decidable for task automata with fixed computation times.*

This theorem follows from Lemma 8, 10, and 11 established in the following section.

The result holds also for task automata with interval computation times in case the finishing time of a task has no influence on the new task releases of the automata and on the decisions of the scheduling strategy (EDF and FPS are such scheduling strategies). In fact, it can be converted to the case of task automata with fixed computation times.

Theorem 3 *The problem of checking schedulability relative to FPS or EDF scheduling strategy for task automata without task feedback is decidable.*

To prove this, we show that if a non-schedulable state is reachable then it is reachable also when the computations of all tasks take the worst-case execution time. This is formulated as Lemma 12 and proved in the following section. This fact together with Theorem 2 proves Theorem 3.

From scheduling theory [1], we know that the preemptive version of the earliest deadline first (EDF) scheduling strategy is optimal in the sense that if a task queue is non-schedulable with EDF, it cannot be schedulable with any other scheduling strategy (preemptive or non-preemptive). Thus, the general schedulability checking problem is equivalent to the relative schedulability checking with respect to EDF.

For decidability, we have a general result that follows from the above theorems.

Theorem 4 *The problem of checking schedulability is decidable for task automata without task feedback or with fixed computation times.*

Unfortunately the schedulability problem is undecidable for preemptive scheduling when tasks may have variable computation times and the finishing time of a task may be used to influence (i.e., feedback on) the behaviour of the automaton. This is stated in the following theorem which is our second main technical result.

Theorem 5 *The problem of checking whether a task automaton is schedulable with FPS is undecidable.*

This result is established in Section 5. However, the proof does not depend on fixed priority scheduling strategy and it can be easily modified for almost all preemptive scheduling strategies (e.g., the proof holds for EDF and SJF without any modification).

4 Decidability

We shall encode the schedulability checking problem for task automata with fixed computation time of tasks as a reachability problem of timed automata with bounded subtraction. We show that a reachability problem for this extension of timed automata is decidable. In the last part of this section, we show that the schedulability checking problem for task automata without task feedback can be reduced to the schedulability checking of task automata with fixed computation time of tasks.

4.1 Timed Automata with Subtraction

We shall identify a class of suspension automata [3] that are timed automata with subtraction in which clocks may be updated by subtraction under certain conditions. We show that if for each clock there is a known maximal constant such that subtraction operations are performed on clocks only in the bounded zone, the reachability problem is decidable. Because the schedulability checking problem can be coded as a reachability problem for such automata, it is decidable.

Definition 8 *A timed automaton with subtraction is a timed automaton in which clocks may be updated by subtraction in the form $x := x - C$ in addition to reset of the form $x := 0$, where C is a natural number.*

This is the class of so-called suspension automata [3], for which it is known that the reachability problem is undecidable. However, for the following class of suspension automata, the location reachability problem is decidable.

Definition 9 *(Timed Automata with Bounded Subtraction) A timed automaton with bounded subtraction is a timed automaton such that there is a constant M_x for each clock x (the ceiling of x), and for all its reachable states (l, u)*

1. $u(x) \geq 0$ for all clocks x , i.e., clock values should not be negative and
2. $u(x) \leq M_x$ if $l \xrightarrow{g^{ar}} l'$ for some l' and C such that $u \models g$ and $(x := x - C) \in r$.

Note that the two conditions imply that in a state (l, u) , a clock x is allowed to be subtracted by a constant C only if $C \leq u(x) \leq M_x$.

Because subtractions on clocks are performed only when clocks are bounded with known constants, they preserve the standard region equivalence [5]. It is illustrated in Figure I.3.

Definition 10 (Region Equivalence \sim [5]) For a clock $x \in \mathcal{C}$, let C_x be a natural number. For a real number t , let $\{t\}$ denote the fractional part of t , and $\lfloor t \rfloor$ denote its integer part. Let $u, v \in \mathcal{V}$. We define $u \sim v$, i.e., u, v are region-equivalent iff

1. for each clock x , either $\lfloor u(x) \rfloor = \lfloor v(x) \rfloor$ or $u(x) > C_x$ and $v(x) > C_x$ and
2. for all clocks x, y if $u(x) \leq C_x$ and $u(y) \leq C_y$ then
 1. $\{u(x)\} = 0$ iff $\{v(x)\} = 0$ and
 2. $\{u(x)\} \leq \{u(y)\}$ iff $\{v(x)\} \leq \{v(y)\}$

However, the standard region construction above deals only with automata containing no diagonal constraints, i.e., bounds on the clock differences. To encode scheduling problems, we need to use diagonal constraints as guards in automata. For example to check the schedulability of a task automaton related to SJF, we need to compare the difference between clocks to decide where to insert a new task. We need the refined version of region equivalence from [6].

Definition 11 (Refined Region Equivalence \approx) Let \mathcal{G} be a finite set of diagonal constraints in the form $x - y \bowtie N$ where N is a natural number. Let $u, v \in \mathcal{V}$. We define $u \approx v$ iff

1. $u \sim v$
2. $u \models g$ iff $v \models g$ for all $g \in \mathcal{G}$

For a clock assignment u , let $u(x - C)$ denote the assignment where the value of x is subtracted by C , namely $u(x - C)(x) = u(x) - C$ and $u(x - C)(y) = u(y)$ for $y \neq x$. The following congruence properties of the refined region equivalence will give rise to a finite partitioning of the reachable state space for a timed automaton with bounded subtraction.

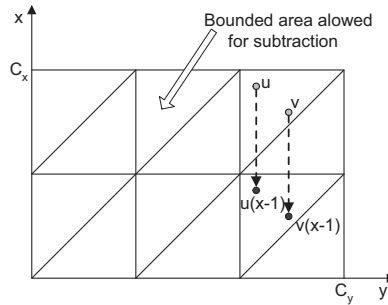


Figure I.3: Region equivalence preserved by subtraction when clocks are bounded.

Lemma 6 Given a timed automaton with bounded subtraction, let \mathcal{G} denote the set of diagonal constraints appearing in the automaton and C_x be the maximum of M_x (the ceiling of x) and all constants appearing in the guards and invariants of the automaton involving clock x . Let $u, v \in \mathcal{V}$ and t be a non-negative real number. Then $u \approx v$ implies

1. $u + t \approx v + t'$ for some real number t' such that $\lfloor t \rfloor = \lfloor t' \rfloor$.
2. $u[x \mapsto 0] \approx v[x \mapsto 0]$ for a clock x ,
3. $u(x - C) \approx v(x - C)$ for all natural numbers C such that $C \leq u(x) \leq C_x$.

Proof. The proof is given in Appendix. ■

The refined region equivalence induces a bisimulation over reachable states of timed automata with bounded subtraction, which can be used to partition the whole state space into a finite number of equivalence classes.

Lemma 7 *Assume a timed automaton with bounded subtraction, a location l and clock assignments u and v . Then $u \approx v$ implies that*

1. *whenever $(l, u) \longrightarrow (l', u')$ then $(l, v) \longrightarrow (l', v')$ for some v' s.t. $u' \approx v'$.*
2. *whenever $(l, v) \longrightarrow (l', v')$ then $(l, u) \longrightarrow (l', u')$ for some u' s.t. $u' \approx v'$.*

Proof. It follows from Lemma 6. ■

The above lemma essentially states that if $u \approx v$ then (l, u) and (l, v) are bisimilar, which implies the following result.

Lemma 8 *The location reachability problem for timed automata with bounded subtraction is decidable if the bound M_x for each clock x is known.*

Proof. From Lemma 7, it follows that for each location l of the automaton, there is a finite number of equivalence classes induced by the bisimulation relation \approx . Because the number of locations of an automaton is finite, the whole state space of an automaton can be partitioned into finite number of such equivalence classes and these equivalence classes can be effectively generated. ■

4.2 Preemptive Schedulers as Timed Automata with Subtraction

We shall first consider automata with fixed computation times. Later on, we show how these results extend to task automata without task feedback, that are automata with tasks whose computation times can be intervals but the completion time of a task has no influence on the new releases of task instances.

Because we consider only tasks with fixed computation times, i.e., $B = W$ for each task type $P(B, W, D)$, in the following, we will talk only about the worst case remaining computation time w_i for each task instance $P_i(b_i, w_i, d_i)$ and W_i as a parameter of the task type.

At any time point, the task queue may contain a sorted list of task instances. To be able to talk about task instances according to their task type, we assume some (any) order on the task types. By p_{ij} we denote a j -th task instance of the i -th task type. Note that both indices are bounded, one by the number of the task types and the other one by the maximal number of task instances of a particular task type in a not overflowed task queue. Indices of task instances of the same type do not correspond to their order in the queue, they just allow us to distinguish between two instances of the same type. They are also reused when a task instance finishes.

Main ideas:

Essentially, the decidability results are achieved by the encoding of task preemptions using subtraction on clocks by constants (instead of using stop-watches). The difficult part is to show how does the automaton keep the information about the remaining computation time and deadline of the task instances in the queue using clocks. For each released task instance, we use two clocks: a computing clock to remember the accumulated computation time the instance has consumed so far, and a deadline clock to remember the remaining relative deadline since release. To deal with preemption, we use subtraction. When the running task is preempted, we do not stop the computing clock as we may do in using stopwatch automata. But when a running task finishes, we subtract the computing clocks (for all preempted instances) with the execution time of the finished task, which is precisely the amount of time that the computing clocks have proceeded since the most recent preemption. Assume that the current running task p_{ij} and the task p_{kl} preempted by the current one have computing clocks x_{ij}^c and x_{kl}^c , and their original fixed computation times are W_i and W_k (known constants) respectively. Then $W_i - v(x_{ij}^c)$ stores precisely the remaining computation time for the running task (v is a clock valuation). The remaining computation time for the preempted task is $W_k - (v(x_{kl}^c) - v(x_{ij}^c))$. Following this reasoning, we may recover the remaining computation times using the clock differences for all preempted tasks according to the ordering they are preempted. For example, the remaining computation time for the task instance p_{mn} preempted by p_{kl} is $W_m - (v(x_{mn}^c) - v(x_{kl}^c))$. Using diagonal constraints, the remaining computation times of the preempted task instances may be compared with the known constant computation time of a newly released task instance. Thus a scheduling strategy deciding where the new task instance should be inserted in the task queue based on remaining computation times, can be encoded as a timed automaton with diagonal constraints and bounded subtraction.

Assume a task automaton A with fixed computation times, and a *preemptive scheduling* strategy Sch . As for the non-preemptive case (Theorem 1), let $E(A)$ denote the timed automaton of A where the action label of each edge is replaced with $release_i$ if its target location is mapped to P_i . For a preemptive scheduling strategy Sch , we construct $E(Sch)$ as a timed automaton extended with subtraction to synchronize with $E(A)$ through the actions $release_i$'s. This automaton encodes the queues from all possible adequate states (the task queue is not overflowed and no deadline is violated) in the locations. The edges between the locations are induced by the scheduling strategy and the rules for the task finishing.

We shall show that the LTS induced by $E(A) \parallel E(Sch)$ for any A and Sch is weakly timed bisimilar to $\llbracket A_{Sch} \rrbracket$, i.e., the LTS induced by A with respect to Sch , restricted to the adequate states. We shall also show how to detect reachability of non-adequate states in $\llbracket A_{Sch} \rrbracket$.

Each location of $E(Sch)$ encodes the information about the discrete part of the queue, containing the names of the types of the task instances and their

status: running, preempted, or just released. We encode this information into variables (with finite domain). For each p_{ij} , we use a state variable $\text{Status}(ij)$ initialized to free, representing the current status of p_{ij} :

- $\text{Status}(ij) = \text{free}$ means that the position i, j in the task queue is free (i.e., p_{ij} is finished or not released yet),
- $\text{Status}(ij) = \text{released}$ means that p_{ij} is released, not started yet,
- $\text{Status}(ij) = \text{running}$ means that p_{ij} is running on the processor, and
- $\text{Status}(ij) = \text{preempted}$ means that p_{ij} has started its computation but it is suspended now.

The remaining computation time w_{ij} of a preempted task p_{ij} ($\text{Status}(ij) = \text{preempted}$) may be a real number; but if p_{ij} is only released ($\text{Status}(ij) = \text{released}$), w_{ij} is an integer W_i . We introduce two clocks for each p_{ij} :

- x_{ij}^c (a computing clock) is used to remember the accumulated computation time since p_{ij} was started.
- x_{ij}^d (a deadline clock) is used to remember the deadline and it is reset to 0 when p_{ij} is released.

Since we cannot stop a computing clock for a preempted task instance p_{ij} , we have to remember which task instance has preempted it (or which task instance is the closest one running instead p_{ij}). The predicate $\text{Preempted_by}(ij)(kl)$ is true if and only if $\text{Status}(ij) = \text{preempted}$, and p_{kl} is the closest task instance in the queue to p_{ij} such that $\text{Status}(kl) = \text{preempted}$ or $\text{Status}(kl) = \text{running}$, and p_{kl} is closer to the head of the queue than p_{ij} .

We will use locations to remember which (if any) task instance is running at the moment. $E(\text{Sch})$ has the following types of locations:

- Idling denotes that the task queue is currently empty, that is, $\text{Status}(ij) = \text{free}$ for all i, j .
- $\text{Running}(ij)$ denotes that a task instance p_{ij} is running, that is, $\text{Status}(ij) = \text{running}$. Each such location has invariants $x_{ij}^c \leq W_i$ and $x_{kl}^d \leq D_k$ for all k, l such that $\text{Status}(kl) \neq \text{free}$.

We also need to keep track of the order of the task instances in $E(\text{Sch})$. For each p_{ij} , we use a state variable $\text{Position}(ij)$ which is set to the value \perp for all i, j such that $\text{Status}(ij) = \text{free}$ and it is set to a natural number denoting the position of p_{ij} in the queue otherwise. If $\text{Status}(ij) = \text{running}$ then $\text{Position}(ij) = 1$. Since the queue size is bounded in all adequate states, $\text{Position}(ij)$ has a finite domain for all p_{ij} (and thus can be encoded into timed automata).

When a task finishes or a new one arrives, we use a predicate $\text{Head}(mn)$ to denote the fact that the task instance p_{mn} is the next task instance to be executed (the next head of the queue). If a new task instance arrives, we take it also into account. Conversely, when a task finishes, we do not consider it anymore.

We need to update the positions of the tasks in the queue when a task instance arrives or finishes. Let $\text{Remove}(mn)$ denote the update of the variables $\text{Position}(ij)$ for all p_{ij} when the task instance p_{mn} is removed from the queue (it has finished). When a new instance p_{mn} arrives, $\text{Insert}(mn)$ denotes the

update of the variables $\text{Position}(ij)$ corresponding to inserting the new task instance into the (discrete part of the) queue according to the scheduling strategy.

According to our definition of scheduling strategies, $\text{Head}(mn)$, $\text{Remove}(mn)$, and $\text{Insert}(mn)$ can be encoded using (diagonal free) timed automata guards on the values w_{ij}, d_{ij} of the task instances. These values can be at any timepoint reconstructed from the clocks x_{ij}^c and x_{ij}^d (as is proved below) and therefore the predicates can be encoded as (diagonal) guards on the edges of $E(\text{Sch})$. We show how this can be done for EDF and SJF in the Appendix (the case of FPS is straightforward, because it uses only the static task parameters).

The edges of $E(\text{Sch})$ between its locations are defined in Table I.1. Later we also add a location Err which will be reachable if and only if either a deadline can be missed or the queue can overflow.

At first, we prove that $E(A) \parallel E(\text{Sch})$ is a faithful translation of A and Sch restricted to the adequate states by showing a weak timed bisimulation between their semantical labeled transition systems.

Lemma 9 *The LTS induced by $E(A) \parallel E(\text{Sch})$ is weak timed bisimilar to $\llbracket A_{\text{Sch}} \rrbracket$ restricted to the adequate states, based on an abstract transition relation which abstracts away from action labels, defined as follows:*

1. $s_A \xrightarrow{\tau} s'_A$ iff $s_A \xrightarrow{a} s'_A$ for an action $a \in \text{Act} \cup \{\text{fin}\}$.

2. $s_A \xrightarrow{t} s'_A$ iff $s_A \xrightarrow{t} s'_A$ for a time delay t .

where s_A, s'_A are both either states of $\llbracket A_{\text{Sch}} \rrbracket$ or states of $E(A) \parallel E(\text{Sch})$.

Proof.

Assume that the task queue in a state (l, u, q) of A , contains triples (b_{ij}, w_{ij}, d_{ij}) where $b_{ij} = w_{ij} \geq 0$ and d_{ij} are the remaining computation time and relative deadline for a task instance p_{ij} (we use the same indexing as in $E(\text{Sch})$). By $p_{ij} <_q p_{kl}$ we denote the fact that $p_{ij} \in q$, $p_{kl} \in q$, and the task instance p_{ij} in q is closer to the head of the queue than the task instance p_{kl} .

We define the following relations:

$$S_1 = \{((l, u, \square), (\langle l, \text{ldling} \rangle, (u \cup v))) \mid l \in N, u, v \in \mathcal{V}\}$$

$$S_2 = \{((l, u, q), (\langle l, \text{Running}(mn) \rangle, (u \cup v))) \mid l \in N, u, v \in V \ C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5\} \text{ where}$$

- $C_1 \equiv [p_{ij} \in q \Leftrightarrow \text{Status}(ij) \neq \text{free} \ \wedge \ p_{ij} <_q p_{kl} \Leftrightarrow \text{Position}(ij) < \text{Position}(kl)]$
- $C_2 \equiv [w_{ij} = W_i - v(x_{ij}^c) \text{ for } i, j \text{ s.t. } \text{Status}(ij) = \text{running}]$
- $C_3 \equiv [w_{ij} = W_i - (v(x_{ij}^c) - v(x_{kl}^c)) \text{ for all } i, j \text{ s.t. } \text{Status}(ij) = \text{preempted} \text{ and } k, l \text{ s.t. } \text{Preempted_by}(ij)(kl)]$
- $C_4 \equiv [w_{ij} = W_i \text{ for all } i, j \text{ s.t. } \text{Status}(ij) = \text{released}]$
- $C_5 \equiv [d_{ij} = D_i - v(x_{ij}^d) \text{ for all } i, j \text{ s.t. } \text{Status}(ij) \neq \text{free}]$

We establish that $S = S_1 \cup S_2$ is a weak timed bisimulation.

First consider a pair in S_1 , $((l, u, \square), (\langle l, \text{ldling} \rangle, (u \cup v)))$.

- If A or $E(A)$ takes a discrete transition which does not release a task instance (is not labeled by a release_i) or a time delay then the other automaton

1. Idling to Running(ij):
 - guard: none
 - action: release_i
 - reset: $x_{ij}^c := 0, x_{ij}^d := 0, \text{Status}(ij) := \text{running}$
2. Running(ij) to Idling:
 - guard: $\text{Status}(kl) = \text{free}$ for all $k, l \neq i, j, x_{ij}^c = W_i$
 - action: none
 - reset: $\text{Status}(ij) := \text{free}, x_{done} := 0, \text{Remove}(ij)$
3. Running(ij) to Running(mn): there are three types of edges.
 1. The running task instance p_{ij} is finished, p_{mn} was preempted and is scheduled to run now:
 - guard: $x_{ij}^c = W_i, \text{Status}(mn) = \text{preempted}, \text{Head}(mn)$
 - action: none
 - reset: $\text{Status}(ij) := \text{free}, x_{done} := 0, x_{kl}^c := x_{kl}^c - W_i$ for all k, l s.t. $\text{Status}(kl) = \text{preempted}, \text{Status}(mn) := \text{running}, \text{Remove}(ij)$
 2. A new task instance p_{mn} is released, which preempts the running task instance p_{ij} :
 - guard: $\text{Status}(mn) = \text{free}, \text{Head}(mn)$
 - action: release_m
 - reset: $\text{Status}(mn) := \text{running}, x_{mn}^c := 0, x_{mn}^d := 0, \text{Status}(ij) := \text{preempted}, \text{Insert}(mn)$
 3. The running task instance p_{ij} is finished and p_{mn} (which was released but has never run) is scheduled to run now:
 - guard: $x_{ij}^c = W_i, \text{Status}(mn) = \text{released}, \text{Head}(mn)$
 - action: none
 - reset: $\text{Status}(ij) := \text{free}, x_{done} := 0, x_{mn}^c := 0, x_{kl}^c := x_{kl}^c - W_i$ for all k, l s.t. $\text{Status}(kl) = \text{preempted}, \text{Status}(mn) := \text{running}, \text{Remove}(ij)$
4. Running(ij) to Running(ij): a new task is released and the running task instance p_{ij} will continue to run:
 - guard: $\text{Status}(mn) = \text{free}, \text{Head}(ij)$
 - action: release_m
 - reset: $\text{Status}(mn) := \text{released}, x_{mn}^d := 0, \text{Insert}(mn)$

Table I.1: *Defining the edges of $E(\text{Sch})$.*

can simply take the same transition. Note, that $E(\text{Sch})$ can delay in Idling for an unbounded amount of time.

- If A takes a discrete transition releasing an instance of P_i then $E(\text{Sch})$ takes the corresponding transition labeled by release_i and moves to $\text{Running}(ij)$ (Rule 1 in Table I.1). It is easy to check that conditions C_1 – C_5 hold, because x_{ij}^c and x_{ij}^d are reset and $w_{ij} = W_i, d_{ij} = D_i$.
- If $E(\text{Sch})$ takes a transition labeled by release_i and moves to $\text{Running}(ij)$ (Rule 1 in Table I.1) then A takes the corresponding discrete transition which releases an instance of P_i .

Now we consider pairs in S_2 and possible transitions according to their type.

Discrete transition. Assume $((l, u, q), (\langle l, \text{Running}(ij) \rangle, (u \cup v))) \in S_2$ and A takes a transition $(l, u, q) \longrightarrow (l', u', \text{Sch}(M(l'), q))$. Further assume that this transition is induced by $l \xrightarrow{g^{ar}} l', u \models g$, and $u[r] \models I(l')$. Then $E(A) \parallel E(\text{Sch})$ takes the corresponding transition. There are three possibilities:

- $M(l') = \perp$ — the corresponding transition in $E(A) \parallel E(\text{Sch})$ is enabled and both resulting states trivially belong to S_2 .
- $M(l') = P_m(B_m, W_m, D_m)$ and the new instance preempts the currently running one, i.e., $\text{Sch}(P_m(B_m, W_m, D_m), q) = P_m(B_m, W_m, D_m) :: q$ — the corresponding transition in $E(A) \parallel E(\text{Sch})$ (Rule 2 in Table I.1) is enabled because there exists n such that $\text{Status}(mn) = \text{free}$ (A is restricted to not overflowed queues) and $\text{Head}(mn)$ is true. Conditions C_1 – C_5 hold.
- $M(l') = P_m(B_m, W_m, D_m)$ and $\text{Sch}(P_m(B_m, W_m, D_m), P_{ij}(b_{ij}, w_{ij}, d_{ij}) :: q) = P_{ij}(b_{ij}, w_{ij}, d_{ij}) :: q'$ — the corresponding transition in $E(A) \parallel E(\text{Sch})$ (Rule 4 in Table I.1) is enabled because there exists n such that $\text{Status}(mn) = \text{free}$ (A is restricted to not overflowed queues) and $\text{Head}(ij)$ is true. Conditions C_1 – C_5 hold in the new pair of states.

The other direction is symmetrical if we notice that $\text{Head}(kl)$ can be true only for either i, j or m, n (and for only one of them).

Time pass. Assume $((l, u, P_{ij}(b_{ij}, w_{ij}, d_{ij}) :: q), (\langle l, \text{Running}(ij) \rangle, (u \cup v))) \in S_2$ and A takes a transition $(l, u, P_{ij}(b_{ij}, w_{ij}, d_{ij}) :: q) \xrightarrow{t} (l, u', \text{Run}(P_{ij}(b_{ij}, w_{ij}, d_{ij}) :: q, t))$. We know that $t \leq w_{ij}$ and $u + t \models I(l)$. Then $E(A) \parallel E(\text{Sch})$ delays for the same amount of time. $E(A)$ can delay for t , $u + t \models I(l)$ is a sufficient condition. For $E(\text{Sch})$, we need to satisfy $(v + t)(x_{ij}^c) \leq W_{ij}$ and $(v + t)(x_{kl}^d) \leq D_{kl}$ for all k, l such that $\text{Status}(kl) \neq \text{free}$. The former holds because of Condition C_2 , the latter because of C_5 and the fact that A is restricted to the states where no deadline is missed. Conditions C_1 – C_5 hold in the new pair of states.

The other direction is symmetrical, Conditions C_1 – C_5 are formulated as equalities.

Finishing a task. Assume $((l, u, P_{ij}(b_{ij}, w_{ij}, d_{ij}) :: q), (\langle l, \text{Running}(ij) \rangle, (u \cup v))) \in S_2$ and A takes a transition $(l, u, P_{ij}(b_{ij}, w_{ij}, d_{ij}) :: q) \xrightarrow{\tau} (l, u[x_{done}], q)$. We know that $w_{ij} = 0$ and $u[\text{done}] \models I(l)$. Because of Condition C_2 , we know that $x_{ij}^c = W_i$. If $q = \square$ then the transition given by Rule 2 in Table I.1 is enabled and $E(A) \parallel E(\text{Sch})$ takes it. Conditions C_1 – C_5 hold. Otherwise, $\text{Head}(mn)$ is true for some (unique) m, n .

If $\text{Status}(mn) = \text{preempted}$ then $E(A) \parallel E(\text{Sch})$ takes the transition given by Rule 1 in Table I.1. Subtraction of clocks in the reset together with Condition C_3 ensure that C_2 and C_3 hold in the new pair of states. The other conditions also hold.

If $\text{Status}(mn) = \text{released}$ then $E(A) \parallel E(\text{Sch})$ takes the transition given by Rule 3 in Table I.1. Subtraction of clocks in the reset together with Condition C_3 ensure that C_3 hold in the new pair of states. The other conditions also hold.

Again, the other direction is symmetrical. \blacksquare

Now we show that the product automaton $E(A) \parallel E(\text{Sch})$ is a bounded timed automaton with subtraction.

Lemma 10 *The automaton $E(A) \parallel E(\text{Sch})$ for a task automaton A and a preemptive scheduling strategy Sch is a timed automaton with bounded subtraction.*

Proof. Note that $x_{kl}^c \leq x_{kl}^d$ because whenever x_{kl}^d is reset to zero, so is x_{kl}^c (when a new instance of P_k is released). Note also that all edges labeled with a subtraction lead from and to a location with the invariant $x_{kl}^d \leq D_k$ for all k, l such that $\text{Status}(kl) \neq \text{free}$. Thus x_{kl}^d is bounded by D_k and therefore x_{kl}^c is bounded by D_k for k, l such that $\text{Status}(kl) \neq \text{free}$. But only such clocks can be subtracted in $E(\text{Sch})$.

Secondly, the only possibility for a computing clock, say x_{kl}^c for a task instance p_{kl} , to become negative is by a subtraction. But a subtraction is done on x_{kl}^c only when a task instance, say p_{ij} , is finished, i.e., $x_{ij}^c = W_i$ holds. Note that $\text{Status}(kl) = \text{preempted}$ implies that p_{kl} was released and started (when $\text{Status}(kl)$ was set to running) before $\text{Status}(ij)$ was set to running. Otherwise, $\text{Status}(kl) = \text{released}$. That is x_{kl}^c is reset to zero before x_{ij}^c . Thus we have $x_{kl}^c \geq x_{ij}^c$ implying that $x_{kl}^c - W_i \geq 0$ when $x_{ij}^c = W_i$. Therefore, all clocks are non-negative. \blacksquare

To detect whether a non-adequate state is reachable in $\llbracket A_{\text{Sch}} \rrbracket$, we add a location Err and the edges to $E(\text{Sch})$ as described in Table I.2.

Now we have the correctness lemma for our encoding. Let v_0 be a clock valuation which assigns zero to all clocks of the scheduler automaton $E(\text{Sch})$.

Lemma 11 *Let A be a task automaton with fixed computation time of tasks and Sch a preemptive scheduling strategy. Assume that (l_0, u_0, \square) and $(\langle l_0, \text{ldling} \rangle, u_0 \cup v_0)$ are the initial states of A and the product automaton $E(A) \parallel E(\text{Sch})$ respectively. Then*

1. *if there are l and u such that $(l_0, u_0, \square) \xrightarrow{*} (l, u, q_{\text{err}})$ then $(\langle l_0, \text{ldling} \rangle, u_0 \cup v_0) \xrightarrow{*} (\langle l', \text{Err} \rangle, u' \cup v)$ for some l', u' , and v , and*
2. *if there are l, u , and v such that $(\langle l_0, \text{ldling} \rangle, u_0 \cup v_0) \xrightarrow{*} (\langle l, \text{Err} \rangle, u \cup v)$ then $(l_0, u_0, \square) \xrightarrow{*} (l', u', q_{\text{err}})$ for some l', u' .*

Proof. Consider a finite path ρ in $\llbracket A_{\text{Sch}} \rrbracket$ leading to an error state (l, u, q_{err}) . Let (l_e, u_e, q_e) denote the last state on the prefix of ρ containing only adequate states.

1. Running(ij) to Err:
 - guard: $x_{ij}^c < W_i, x_{ij}^d = D_i$
 - action: none
 - reset: none
2. Running(ij) to Err: for each k, l there is an edge labeled with
 - guard: Status(kl) = released, $x_{kl}^d > D_k - W_k$
 - action: none
 - reset: none
3. Running(ij) to Err: for each k, l there is an edge labeled with
 - guard: Status(kl) = preempted, $x_{ij}^c < W_i, x_{kl}^d = D_k$
 - action: none
 - reset: none
4. Running(ij) to Err: for each k there is an edge labeled with
 - guard: Status(kl) \neq free for all l
 - action: release $_k$
 - reset: none

Table I.2: *Defining the edges of $E(\text{Sch})$ leading to Err.*

If a deadline will be missed in the immediate successor of (l_e, u_e, q_e) then there must be a task instance $P_{kl}(b_{kl}, w_{kl}, d_{kl}) \in q_e$ such that $d_{kl} = 0$ and $w_{kl} > 0$. According to Lemma 9, a corresponding state in $E(A) \parallel E(\text{Sch})$ is reachable. Because of Conditions C_1 – C_5 , one of the transitions given by Rules 1–3 in Table I.2 is enabled according to the status of $P_{kl}(b_{kl}, w_{kl}, d_{kl})$. If the queue will overflow in the immediate successor of (l_e, u_e, q_e) then a transition given by Rule 4 in Table I.2 is enabled because of C_1 .

If an error state is reachable in the LTS induced by $E(A) \parallel E(\text{Sch})$ then it is either by taking a transition given by Rules 1–3 in Table I.2 or by the transition given by Rule 4 in Table I.2. In the former case, there must be a task instance $P_{kl}(b_{kl}, w_{kl}, d_{kl})$ such that $d_{kl} = 0$ and $w_{kl} > 0$ in the task queue according to Lemma 9. In the latter case, the queue overflows. Because the time does not stop in A , a deadline will be missed in both cases. ■

The above lemma states that the schedulability analysis problem can be solved by reachability analysis for timed automata extended with subtraction. From Lemma 10, we know that $E(\text{Sch})$ is bounded. Because the reachability problem is decidable due to Lemma 8, we complete the proof for our main result stated in Theorem 2.

4.3 Task Automata without Task Feedback

Now we prove that a task automaton without task feedback is schedulable with a preemptive scheduling strategy which is either FPS or EDF if and only if it is schedulable (with the same scheduling strategy) when the tasks have constant computation times equal to the given worst-case computation times.

Lemma 12 *Let A be a task automaton without task feedback with an initial state (l_0, u_0, \square) and Sch be either FPS or EDF. If there is a path $r_1 = (l_0, u_0, \square) \xrightarrow{*} (l, u, q_{err})$ in $\llbracket A_{Sch} \rrbracket$ then there is a path $r_2 = (l_0, u_0, \square) \xrightarrow{*} (l', u', q_{err})$ in $\llbracket A_{Sch} \rrbracket$ such that each task instance $P(b, w, d)$ finishes when $w = 0$.*

Proof. Given a path r_1 , we construct r_2 so that we let the underlying timed automaton of A to perform the same transitions as in r_1 (discrete transitions are taken at the same absolute time points) and all task instances $P(b, w, d)$ finish when $w = 0$ (task finishing transitions are taken at possibly different absolute time points). We have to show that r_2 is a path in $\llbracket A_{Sch} \rrbracket$ and that it leads to a state where a deadline is missed.

Because A has no task feedback, the differences in the absolute time points at which task finishing transitions are taken cannot influence the underlying timed automaton of A . In particular, validity of the invariants in the locations is not influenced and all transitions that were enabled along r_1 are also enabled along r_2 (at the same time points). Also, task instances cannot be forced to finish before $w = 0$. Therefore, r_2 is a path in $\llbracket A_{Sch} \rrbracket$.

To show that r_2 leads to a state where a deadline is missed we show the following proposition true. Let (l, u, q_1) and (l, u, q_2) denote the states of A at the same absolute time point along r_1 and r_2 , respectively. Then q_1 can be obtained from q_2 by dropping some task instances and possibly decreasing some remaining computation times (all remaining deadlines are the same for the task instances present in both queues).

The proof is done by induction on the length of the path. The proposition trivially holds for a path of zero length.

Assume that the proposition holds in states (l, u, q_1) and (l, u, q_2) . If A takes a discrete transition not releasing any task then the proposition holds trivially. If A takes a discrete transition which releases a task instance $P_{ij}(B_{ij}, W_{ij}, D_{ij})$ then we claim that the scheduling strategy inserts this task into the same position in q_1 and q_2 with respect to the tasks that are in both queues. This clearly holds for FPS, because FPS decides only according to the discrete part of the queue. EDF decides just according to the ordering of the remaining deadlines which depend only on the release times of task instances. But release times are the same for both r_1 and r_2 . In other words, if a task instance $Q(b, w, d)$ has the earliest deadline in q_2 and it is still in q_1 then it also has the earliest deadline there. Thus, the proposition holds after releasing $P_{ij}(B_{ij}, W_{ij}, D_{ij})$.

If A delays for a non-zero time then the remaining computation times of both running task instances in q_1 and q_2 is decreased by the same amount. This cannot invalidate the property if the running task instances are the same. Otherwise, according to the induction hypothesis there is a corresponding task instance $Q(b, w, d)$ in q_2 to the running task instance $Q(b', w', d)$ in r_1 . Then $w > w'$ after the time pass transition.

If A finishes a task instance in r_1 then the proposition holds. If A finishes a task instance $P(b, w, d)$ in r_2 then we know that $w = 0$ and from induction

hypothesis either the corresponding task instance $P(b', w', d)$ in r_1 has already finished or $w' = w$ and therefore it must finish at the same time point. ■

As a consequence, we may consider only the worst case computation time of all tasks (all best case computation times are equal to the worst case computation times). That is, the best case execution times have no effect on the schedulability of systems without task feedback. Therefore, Lemma 12 and Theorem 2 prove Theorem 3. Note that Lemma 12 does not hold for all scheduling strategies. For example, it does not hold for SJF.

5 Undecidability

In this section we show that the schedulability problem for task automata in general (i.e., with preemption, task feedback, and variable execution times of tasks) with fixed priority scheduling is undecidable.

The proof is done by reduction of the halting problem for two-counter machine to the schedulability problem for task automata. A *two-counter machine* consists of a finite state control unit and two unbounded non-negative integer counters. Initially, both counters contain the value 0. Such a machine can execute three types of instructions: incrementation of a counter, decrementation of a counter, and branching based upon whether a specific counter contains the value 0. Note that decrementation of a counter with the value 0 leaves this counter unchanged. After execution of an instruction, a machine changes deterministically its state. One state of a two-counter machine is distinguished as *halt state*. A machine halts if and only if it reaches this state.

We present an encoding of a two-counter machine M using a task automaton A_M such that M halts if and only if A_M is non-schedulable, based on the undecidability proofs of [7]. In the construction, the states of M correspond to specific locations of A_M and each counter is encoded by a clock. We show how to simulate the two-counter machine operations. First, we adopt the notion of wrapping of [7].

Definition 12 A task automaton over set of clocks \mathcal{C} is *N-wrapping* if for all states (l, u, q) reachable from its initial state and for all clocks $x \in \mathcal{C}$: $u \models x \leq N$. An *N-wrapping edge* for a clock x and a location l is an edge from l to itself that is labeled with the guard $x = N$ and which resets the clock x . A clock that is reset only by wrapping edges is called *system clock*.⁵ Each time period between two consecutive time points at which any system clock contains value 0 is called *N-wrapping period*.

We use wrapping to simulate discrete steps of a two-counter machine. Each step is modeled by several N-wrapping periods. We define the wrapping-value of a clock to be the value of the clock when the system clock is 0. Note that a clock is carrying the same wrapping value if it is not reset by another edge than the wrapping edges. This principle is shown in Figure I.4, where x_{sys} is

⁵Note that all system clocks contain the same value.

a system clock and clock x_{copy} contains the same wrapping-value when the automaton takes transitions e_1 and e_3 .

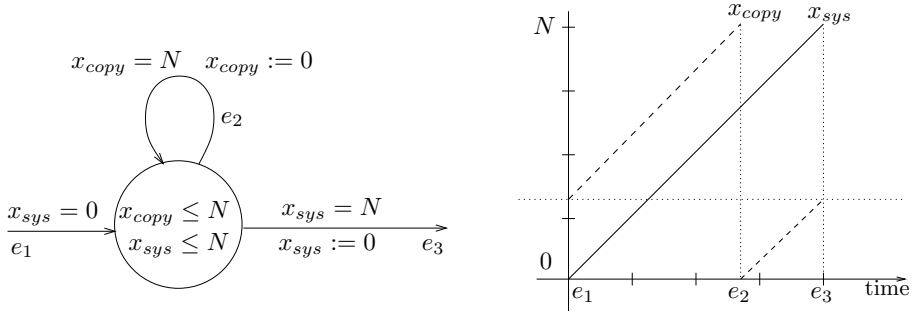


Figure 1.4: The wrapping edge e_2 makes clock x_{copy} carry the same wrapping-value when the transitions e_1 and e_3 are taken.

We encode a two-counter machine M with counters C and D using a 4-wrapping automaton A_M with one system clock denoted x_{sys} and five other clocks $x_c, x_d, x_{old}, x_{copy}$, and x_{done} . In particular, we encode counters C and D of M by clocks x_c and x_d like this: counter value v corresponds to the clock wrapping-value 2^{1-v} . We use the density of the continuous domain to encode arbitrarily large values of the counters. Decrementation (incrementation) of a counter corresponds to doubling (halving) the wrapping-value of the corresponding clock. Test for zero corresponds to the check whether the clock wrapping-value equals to 2.

Now we show how to simulate the decrementation operation by doubling the wrapping-value of the clock x_d . To do this, we use two tasks: P and Q . The task P has execution time in $[0, 1]$ and deadline 50; the task Q has execution time in $[8, 8]$ and deadline 100. Moreover, the priority of P is higher than the priority of Q , i.e., P always preempts Q . Notice that the execution time of task P can vary and the execution time of the task Q is fixed.

The basic idea of doubling a wrapping-value $v \in (0, 1]$ of clock x_d is as follows: we assume that the current wrapping-value of x_d is v . We copy it to the clock x_{copy} (that is, to make the wrapping-value of the clock x_{copy} to be v). Then we release the task Q non-deterministically and reset x_d . The idea is to start Q $2v$ time units before the reset of the system clock x_{sys} and to use x_d to record the response time of Q . Two instances of P are released before Q finishes, that is P preempts Q twice. We make sure that the execution time of each of these two instances of P is exactly v time units. Note that v can be any real number within the interval $(0, 1]$. Then the response time for Q is exactly $8 + 2v$. If Q finishes at a time point when the system clock x_{sys} is reset to 0, the wrapping-value of x_d is $2v$. As Q is released non-deterministically, it is enough if there is one such computation.

To simplify the presentation, we construct A_M with timesteps. But it is easy to see that we could add an unguarded transition into a sink location outgoing

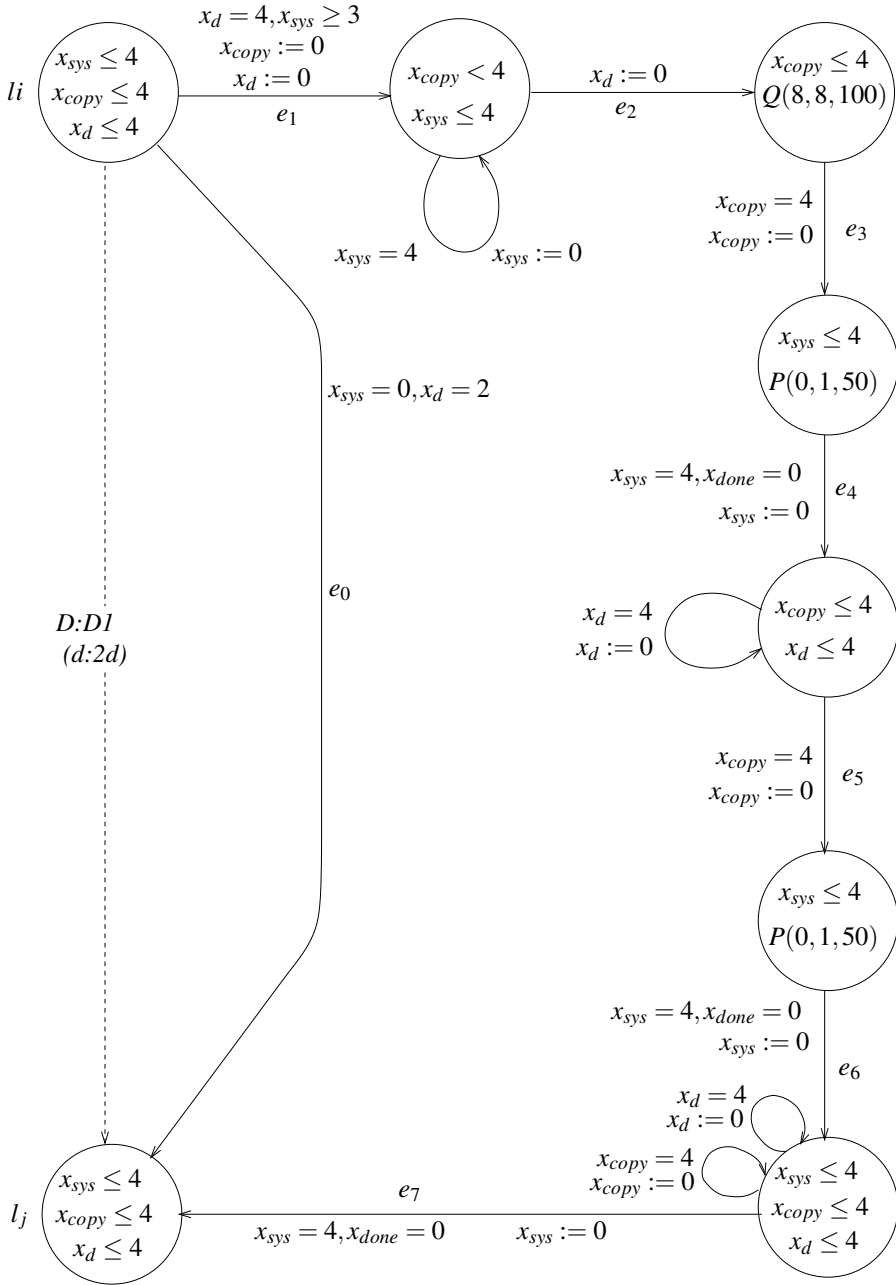


Figure 1.5: A part of reduction automaton corresponding to a decrementation of D . The wrapping edges for clocks x_c, x_{old}, x_{done} , and for all clocks in locations l_i, l_j are omitted. The location invariants $x_c \leq 4, x_{old} \leq 4$, and $x_{done} \leq 4$ are also omitted as well as transitions preventing timesteps.

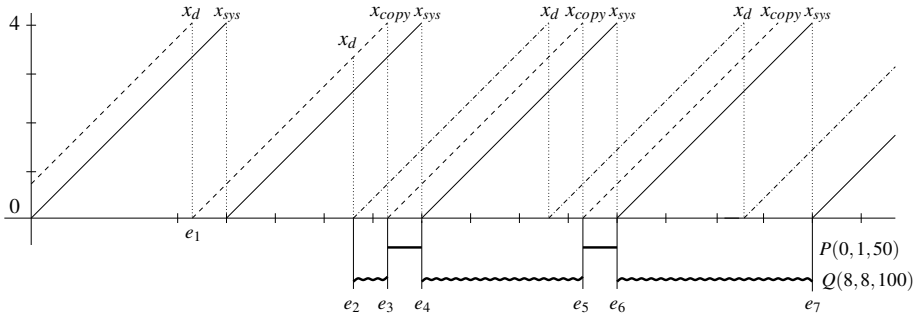


Figure I.6: Time chart of the doubling procedure.

from every location. The sink location does not release any task, it does not have any invariant and it does not have any outgoing transition. A computation leading into this location does not correspond to a computation of the two-counter machine and it does not lead into an error state (a deadline miss). Therefore, it does not influence the correctness of the reduction.

In Figure I.5, we show the part of A_M that doubles the wrapping-value of the clock x_d . Figure I.6 illustrates the time chart of the doubling process. Assume that a two-counter machine M is currently in a state s_i and that it wants to decrease the counter D and then move to a state s_j . The locations l_i and l_j of A_M correspond to the states s_i and s_j respectively. Note that the dashed edge shows the transition of the two-counter machine (it is not a transition of A_M). Note also that the decrementation operation leaves a counter with value 0 unchanged; the automaton can move from l_i directly to l_j through the transition e_0 when x_d contains the wrapping-value 2 (which corresponds to the counter value 0). Otherwise, the following steps are taken to double the wrapping-value of x_d .

First, the wrapping-value of x_d is copied to the clock x_{copy} (by transition e_1), that is, x_{copy} carries the same wrapping-value as x_d . Then the automaton non-deterministically guesses the doubled wrapping-value of x_d (note that when x_d is reset, it will carry a new wrapping-value). It resets x_d at a non-deterministically chosen time instant and at the same time it releases the task Q (transition e_2).

The automaton waits until the clock x_{copy} reaches time 4, then resets x_{copy} and releases P (transition e_3), which preempts Q . Note that the wrapping-value of x_{copy} will remain to be v and at this time point the value of the system clock x_{sys} is $4 - v$. Therefore, x_{sys} will reach 4 in v time units.

The next transition e_4 is guarded by two constraints: $x_{sys} = 4, x_{done} = 0$. To satisfy these constraints, the automaton has to wait in this location for v time units, and the task P must finish at this time point, which resets the clock x_{done} .⁶ By this we make P run (and prevent Q from running) exactly for v time

⁶We have to make sure that x_{done} is not reset by a wrapping edge when it is tested by a guard of the automaton. This causes no technical difficulties and it is omitted from Figure I.5.

units. Now we repeat this procedure again. That is, the automaton waits until $x_{copy} = 4$. Then it releases the task P and forces it to run exactly for v time units (transitions e_5 and e_6).

Now, if the non-deterministic guess of the doubled wrapping-value of x_d was correct, task Q must finish when x_{sys} is equal to 4, which makes the guard on e_7 become true and the automaton moves to location l_j . Thus, if the location l_j is reachable, the wrapping-value of x_d is $2v$. This is stated in the following lemma.

Lemma 13 *Let (l_i, u, q) be an arbitrary state of the automaton shown in Figure I.5 where $u(x_d) = v$, $v \in (0, 1]$, and $q = \square$. Then (l_j, u', q') is reachable for some u' and q' , and if (l_j, u', q') is reachable, it must be the case that $q' = \square$, and $u'(x_d) = 2v$.*

Proof. The proof is obvious from the construction in Figure I.5. ■

To increment a counter we need to halve a wrapping-value of a clock, say x_c . For this, we use the clock x_{old} to copy the wrapping-value of x_c . The new wrapping-value v of x_c is non-deterministically guessed and it is checked by the above doubling procedure. If the wrapping-value of x_{old} (the original wrapping-value of x_c) is $2v$, then the automaton can proceed to the location corresponding to the destination state in an increment instruction.

To simulate branching, we construct two transitions outgoing from a location with guards $x_{sys} = 0 \wedge x_c = 2$ and $x_{sys} = 0 \wedge x_c \neq 2$. The initial state of M corresponds to a location where both x_c and x_d contain the wrapping-value 2. This can be achieved by integer guards and resets.

The halt state corresponds to the location *halt* with unguarded self-loop releasing the task Q whenever it is visited. It follows that the automaton A_M is schedulable if and only if the location *halt* is unreachable, i.e., the two-counter machine M does not halt.

6 Conclusions and Related Work

We have developed a theory of *task automata*, an extended version of timed automata with asynchronous processes, i.e., computation tasks triggered by timed events, which may serve as a model for real time systems with non-uniformly recurring tasks. The model can be used to specify resource requirements and hard timing constraints on computations, in addition to features offered by timed automata. It is general and expressive enough to describe concurrency, synchronization, and tasks which may be periodic, sporadic, preemptive, and (or) non-preemptive as well as dependent relations between tasks. The classical notion of schedulability is naturally extended to the model of task automata.

Our main technical contributions include the proof that the schedulability checking problem related to preemptive scheduling is decidable for a large class of task automata. The problem has been suspected to be undecidable

due to the nature of preemptive scheduling. To our knowledge, this is the first decidability result for preemptive scheduling in dense time models. We believe that our work is one step forward to bridge scheduling theory and automata-theoretic approaches to system modeling and analysis.

The negative result on task automata is that the schedulability checking problem is undecidable for the class of automata with tasks whose computation times are intervals and the completion time of a task may influence the new task releases. We have studied the borderline between decidable and undecidable cases. It is shown that the schedulability problem for many scheduling strategies will be undecidable if the following three conditions hold at the same time: (1) the execution times of tasks are intervals, (2) the precise finishing time of a task may influence the new task releases, and (3) a task is allowed to preempt another running task.

A challenge is to make the results an applicable technique combined with classical methods such as rate monotonic scheduling. We need new algorithms and data structures to represent and manipulate the dynamic task queue consisting of time and resource constraints. As another direction of future work, we shall study the schedule synthesis problem. More precisely given an automaton, it is desirable to characterize the set of schedulable traces accepted by the automaton.

Related work.

This paper summarises and extends our previous results on solving scheduling problems using timed automata; it is the full and extended version of two conference papers. The decidability results have been presented in [8] and the undecidability results in [9]. The general encoding scheme of scheduling strategies has been implemented in the TIMES tool [10], based on the UPPAAL DBM library extended with a subtraction operation. A more recent result shows that for systems where tasks are assigned fixed priorities, the schedulability analysis problem can be solved more efficiently. An encoding of a FPS scheduler using two clocks is presented in [11].

Scheduling is a well-established area. Various analysis methods have been published in the literature. For systems restricted to periodic tasks, algorithms such as rate monotonic scheduling are widely used and efficient methods for schedulability checking exist, see, e.g., [1]. These techniques can be used to handle non-periodic tasks. The standard way is to consider non-periodic tasks as periodic using the estimated *minimal* inter-arrival times as *task periods*. Clearly, the analysis based on such a task model would be pessimistic in many cases, e.g., a task set which is schedulable may be considered as non-schedulable as the inter-arrival times of the tasks may vary over time, that are not necessary minimal. Our work is more related to work on timed systems and scheduling.

An interesting work on relating classical scheduling theory to timed systems is the controller synthesis approach [12, 13]. The idea is to achieve schedulability by construction. A general framework to characterize

scheduling constraints as invariants and synthesize scheduled systems by decomposition of constraints is presented in [13]. However, algorithmic aspects are not discussed in these work. Timed automata has been used to solve non-preemptive scheduling problems mainly for job-shop scheduling [14, 15, 16]. These techniques specify predefined locations of an automaton as goals to achieve by scheduling and use reachability analysis to construct traces leading to the goal locations. The traces are used as schedules. There have been several work, e.g., [3, 17, 18] on using stopwatch automata to model preemptive scheduling problems. As the reachability analysis problem for stopwatch automata is undecidable in general [19], there is no guarantee for termination for the analysis without the assumption that task preemptions occur only at integer points. The idea of subtractions on timers with integers, was first proposed by McManis and Varaiya in [3]. In general, the class of timed automata with subtractions is undecidable, which is shown in [20]. In this paper, we have identified a decidable class of updatable automata, which is precisely what we need to solve scheduling problems without assuming that preemptions occur only at integer points.

Acknowledgement: The work is partially supported by EU through the CREDO project.

References

- [1] G. C. Buttazzo, *Hard Real-Time Computing Systems. Predictable Scheduling Algorithms and Applications*, Kluwer Academic Publishers, 1997.
- [2] C. Ericsson, A. Wall, W. Yi, Timed automata as task models for event-driven systems, in: *Proc. of the 6th International Conference on Real-Time Computing Systems and Applications*, IEEE Computer Society Press, 1999.
- [3] J. McManis, P. Varaiya, Suspension automata: a decidable class of hybrid automata, in: *Proc. of the 6th International Conference on Computer-Aided Verification*, no. 818 in *Lecture Notes in Computer Science*, Springer-Verlag, 1994, pp. 105–117.
- [4] K. G. Larsen, P. Pettersson, W. Yi, Compositional and symbolic model-checking of real-time systems, in: *Proc. of 16th IEEE Real-Time Systems Symposium*, IEEE Computer Society Press, 1995, pp. 76–89.
- [5] R. Alur, D. L. Dill, A theory of timed automata, *Theoretical Computer Science* 126 (2) (1994) 183–235.
- [6] J. Bengtsson, W. Yi, Timed automata: Semantics, algorithms and tools, in: W. Reisig, G. Rozenberg (Eds.), *In Lecture Notes on Concurrency and Petri Nets*, no. 3098 in *Lecture Notes in Computer Science*, Springer-Verlag, 2004.
- [7] T. Henzinger, P. Kopke, A. Puri, P. Varaiya, What’s decidable about hybrid automata?, *Journal of Computer and System Sciences* 57 (1998) 94–124.
- [8] E. Fersman, P. Pettersson, W. Yi, Timed automata with asynchronous processes: Schedulability and decidability, in: J.-P. Katoen, P. Stevens (Eds.), *Proc. of the 8th International Conference on Tools and Algorithms for the Construction and*

- Analysis of Systems, no. 2280 in Lecture Notes in Computer Science, Springer-Verlag, 2002, pp. 67–82.
- [9] P. Krčál, W. Yi, Decidable and undecidable problems in schedulability analysis using timed automata, in: K. Jensen, A. Podelski (Eds.), Proc. of TACAS'04, Barcelona, Spain., Vol. 2988 of Lecture Notes in Computer Science, Springer-Verlag, 2004, pp. 236–250.
- [10] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, W. Yi, TIMES - a tool for modelling and implementation of embedded systems, in: Proc. of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems, no. 2280 in Lecture Notes in Computer Science, Springer-Verlag, 2002.
- [11] E. Fersman, L. Mokrushin, P. Pettersson, W. Yi, Schedulability analysis of fixed-priority systems using timed automata., Theor. Comput. Sci. 354 (2) (2006) 301–317.
- [12] K. Altisen, G. Göbller, A. Pnueli, J. Sifakis, S. Tripakis, S. Yovine, A framework for scheduler synthesis, in: Proc. of the 20th IEEE Real-Time Systems Symposium, IEEE Computer Society Press, 1999, pp. 154–163.
- [13] K. Altisen, G. Göbller, J. Sifakis, A methodology for the construction of scheduled systems, in: Proc. of Formal Techniques in Real-Time and Fault Tolerant Systems, no. 1926 in Lecture Notes in Computer Science, Springer-Verlag, 2000, pp. 106–120.
- [14] Y. Abdeddaim, O. Maler, Job-shop scheduling using timed automata, in: Proc. of 13th Conference on Computer Aided Verification, no. 2102 in Lecture Notes in Computer Science, Springer-Verlag, 2001.
- [15] A. Fehnker, Scheduling a steel plant with timed automata, in: Proc. of the 6th International Conference on Real-Time Computing Systems and Applications, IEEE Computer Society Press, 1999.
- [16] T. Hune, K. G. Larsen, P. Pettersson, Guided Synthesis of Control Programs using UPPAAL, Nordic Journal of Computing 8 (1) (2001) 43–64.
- [17] J. Corbett, Modeling and analysis of real-time ada tasking programs, in: Proc. of 15th IEEE Real-Time Systems Symposium, IEEE Computer Society Press, 1994, pp. 132–141.
- [18] F. Cassez, F. Laroussinie, Model-checking for hybrid systems by quotienting and constraints solving, in: Proc. of the 12th International Conference on Computer-Aided Verification, no. 1855 in Lecture Notes in Computer Science, Springer-Verlag, 2000, pp. 373–388.
- [19] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, S. Yovine, The algorithmic analysis of hybrid systems, Theoretical Computer Science 138 (1) (1995) 3–34.
- [20] P. Bouyer, C. Dufourd, E. Fleury, A. Petit, Are timed automata updatable?, in: Proc. of the 12th International Conference on Computer-Aided Verification, Vol. 1855 of Lecture Notes in Computer Science, Springer-Verlag, 2000.
- [21] K. G. Larsen, W. Yi, Time-abstracted bisimulation: Implicit specifications and decidability, Information and Computation 134 (2) (1997) 75–101.

7 Appendix

Proof of Lemma 6.

Assume $u \approx v$. To prove the first two clauses, we use the known fact on the standard region equivalence \sim , that $u \sim v$ implies that for all t , $u + t \sim v + t'$ for some real number t' such that $\lfloor t \rfloor = \lfloor t' \rfloor$ and $u[x \mapsto 0] \sim v[x \mapsto 0]$ for a clock x . Proofs can be found in the literature, e.g., [21]. Assume $g \in \mathcal{G}$ and g is in the form $x - y \bowtie N$. We have two cases:

1. First, assume $u + t \models g$, that is, $u(x + t) - u(y + t) \bowtie N$. This implies $u(x) - u(y) \bowtie N$. Thus, $u \models g$. Because $u \approx v$, we also have $v \models g$. As $v \models g$ implies $v(x + t) - v(y + t) \bowtie N$ for any real t , we have $v + t' \models g$.
2. Second, assume $u[x \mapsto 0] \models g$, that is, $-u(y) \bowtie n$. As $u \sim v$, $-v(y) \bowtie N$ by definition of \sim . That is, $v[x \mapsto 0] \models g$. The case for $u[y \mapsto 0] \models g$ is similar. Therefore, we conclude the first two clauses in the lemma.

Now we prove the third clause. Assume that $u \sim v$, and for each clock x , $C \leq u(x) \leq C_x$. From $u \sim v$, we have $C \leq v(x) \leq C_x$. We have three cases to check:

1. (The integer parts of clock values in $u(x - C)$ and $v(x - C)$). Because $u \sim v$, we have $\lfloor u(x) \rfloor = \lfloor v(x) \rfloor$. As $C \leq u(x) \leq C_x$, $\lfloor u(x - C)(x) \rfloor = \lfloor v(x - C)(x) \rfloor$. By definition, we have $\lfloor u(x - C)(y) \rfloor = \lfloor v(x - C)(y) \rfloor$ for all clocks y . This proves that the integer parts of clock values in $u(x - C)$ and $v(x - C)$ are equal.
2. (The fractional parts of clock values in $u(x - C)$ and $v(x - C)$). As the subtraction operation on a clock only changes the integer part of the clock, we have, for all clocks y and z ,
 1. $\{u(x - C)(y)\} = 0$ iff $\{v(x - C)(y)\} = 0$ and
 2. $\{u(x - C)(y)\} \leq \{u(x - C)(z)\}$ iff $\{v(x - C)(y)\} \leq \{v(x - C)(z)\}$
3. (The diagonal constraints). Assume that g is in the form $x - y \bowtie N$ and $u(x - C) \models g$, i.e., $u(x) - C - u(y) \bowtie N$. We need to establish that $v(x) - C - v(y) \bowtie N$.

Let $u(x) = \lfloor u(x) \rfloor + \{u(x)\}$, $u(y) = \lfloor u(y) \rfloor + \{u(y)\}$ and $M = \lfloor u(x) \rfloor - \lfloor u(y) \rfloor$. As $u \sim v$, $C \leq u(x) \leq C_x$ and $C \leq v(x) \leq C_x$, we have $\lfloor v(x) \rfloor - \lfloor v(y) \rfloor = M$. Now we need to prove that $\{u(x)\} - \{u(y)\} \bowtie N + C - M$ implies $\{v(x)\} - \{v(y)\} \bowtie N + C - M$.

Consider the three cases of \bowtie :

1. $\{u(x)\} - \{u(y)\} = N + C - M$. It must be the case that $N + C - M = 0$. By $u \sim v$, $\{v(x)\} - \{v(y)\} = 0$.
2. $\{u(x)\} - \{u(y)\} < N + C - M$. As $N + C - M$ is an integer and $\{u(x)\} - \{u(y)\}$ is a real number within the open interval $(-1, 1)$, we must have $N + C - M \geq 0$. In case $N + C - M = 0$, we have $\{u(x)\} - \{u(y)\} < 0$ which implies $\{v(x)\} - \{v(y)\} < 0$ from $u \sim v$. In case $N + C - M > 0$, we have immediately $\{v(x)\} - \{v(y)\} < N + C - M$.
3. $\{u(x)\} - \{u(y)\} > N + C - M$. It is to prove $\{v(y)\} - \{v(x)\} < -N - C + M$ under the assumption $\{u(y)\} - \{u(x)\} < -N - C + M$. This is similar to the above case. ■

Example encoding of scheduling policies.

For EDF, we need to compare the remaining relative deadlines. This information is kept in the clocks x_{ij}^d for all i, j such that $\text{Status}(ij) \neq \text{free}$. We assume that EDF handles the task instances with the same remaining deadline in the FIFO order. Let us denote the first task instance in the queue as p_{ef} and the second one as p_{gh} . Then

- $\text{Head}(gh)$ is true if p_{ef} finishes its computation,
- $\text{Head}(ef)$ is true if p_{mn} is released and $x_{ef}^d \geq D_e - D_m$,
- $\text{Head}(mn)$ is true if p_{mn} is released and $x_{ef}^d < D_e - D_m$, and
- $\text{Head}(mn)$ is false otherwise.

$\text{Remove}(ef)$ simply removes the first task instance from the queue when p_{ef} finishes. When a new instance p_{mn} arrives, it is inserted by $\text{Insert}(mn)$ at the head of the queue if $\text{Head}(mn)$, at the tail of the queue if $x_{kl}^d \geq D_k - D_m$ for all k, l such that $\text{Status}(kl) \neq \text{free}$, and between the neighbouring task instances p_{ij} and p_{kl} if $x_{kl}^d \geq D_k - D_m$ and $x_{kl}^d < D_i - D_m$.

Note that $\text{Head}(kl)$ is true for exactly one pair k, l and that $\text{Head}(kl)$, $\text{Remove}(kl)$, and $\text{Insert}(kl)$ correspond to the EDF scheduling policy if the value of the clocks x_{ij}^d is equal to $D_i - d_{ij}$ for each task instance p_{ij} .

For SJF, we need to compare the remaining (worst case) computation times. This information is kept in the clocks x_{ij}^c . We assume that SJF handles the task instances with the same remaining computation time in the FIFO order. Let us denote the first task instance in the queue as p_{ef} and the second one as p_{gh} . Then

- $\text{Head}(gh)$ is true if p_{ef} finishes its computation,
- $\text{Head}(ef)$ is true if p_{mn} is released and $x_{ef}^c \geq W_e - W_m$,
- $\text{Head}(mn)$ is true if p_{mn} is released and $x_{ef}^c < W_e - W_m$, and
- $\text{Head}(mn)$ is false otherwise.

$\text{Remove}(ef)$ simply removes the first task instance from the queue when p_{ef} finishes. When a new instance p_{mn} arrives, it is inserted by $\text{Insert}(mn)$ at the head of the queue if $\text{Head}(mn)$, at the tail of the queue if $W_m \geq W_k$ for all k, l such that $\text{Status}(kl) = \text{released}$ and $W_m - W_k \geq x_{ef}^c - x_{kl}^c$ for all k, l such that $\text{Status}(kl) = \text{preempted}$. It is inserted between the neighbouring task instances p_{ij} and p_{kl} if

- $W_m \geq W_i$ and $W_m < W_k$, where $\text{Status}(ij) = \text{Status}(kl) = \text{released}$,
- $W_m - W_i \geq x_{ef}^c - x_{ij}^c$ and $W_m < W_k$, where $\text{Status}(ij) = \text{preempted}$, $\text{Status}(kl) = \text{released}$, and $\text{Preempted_by}(ij)(ef)$,
- $W_m - W_i \geq x_{ef}^c - x_{ij}^c$ and $W_m - W_k < x_{ij}^c - x_{kl}^c$, where $\text{Status}(ij) = \text{Status}(kl) = \text{preempted}$ and $\text{Preempted_by}(ij)(ef)$,
- $x_{ij}^c \geq W_e - W_m$ and $W_m < W_k$, where $\text{Status}(ij) = \text{running}$ and $\text{Status}(kl) = \text{released}$, or
- $x_{ij}^c \geq W_e - W_m$ and $W_m - W_k < x_{ij}^c - x_{kl}^c$, where $\text{Status}(ij) = \text{running}$ and $\text{Status}(kl) = \text{preempted}$.

Note that $\text{Head}(kl)$ is true for exactly one pair k, l and that $\text{Head}(kl)$, $\text{Remove}(kl)$, and $\text{Insert}(kl)$ correspond to SJF if $w_{ij} = W_i - v(x_{ij}^c)$ for each

running task instance p_{ij} and $w_{ij} = W_i - (v(x_{ij}^c) - v(x_{kl}^c))$ for all preempted task instances p_{ij} where $\text{Preempted_by}(ij)(kl)$].

Paper II



R-automata*

Parosh Aziz Abdulla, Pavel Krcal, and Wang Yi

Department of Information Technology

Uppsala University, Sweden

Email: {parosh,pavelk,yi}@it.uu.se

Abstract. We introduce *R-automata* – a model for analysis of systems with resources which are consumed in small parts but which can be replenished at once. An R-automaton is a finite state machine which operates on a finite number of unbounded counters (modeling the resources). The values of the counters can be incremented, reset to zero, or left unchanged along the transitions. We define the language accepted by an R-automaton relative to a natural number D as the set of words allowing a run along which no counter value exceeds D . As the main result, we show decidability of the universality problem, i.e., the problem whether there is a number D such that the corresponding language is universal. The decidability proof is based on a reformulation of the problem in the language of finite monoids and solving it using the factorization forest theorem. This approach extends the way in which the factorization forest theorem was used to solve the limitedness problem for distance automata in [Sim94]. We also show decidability of the non-emptiness problem and the limitedness problem, i.e., whether there is a natural number D such that the corresponding language is non-empty resp. all the accepted words can also be accepted with counter values smaller than D . Finally, we extend the decidability results to R-automata with Büchi acceptance conditions.

1 Introduction

We consider systems operating on resources which are consumed in small parts and which can be (or have to be) replenished completely at once. To model such systems, we introduce *R-automata* – finite state machines extended by a finite number of unbounded counters corresponding to the resources. The counters can be incremented, reset to zero, or left unchanged along the transitions. When the value of a counter is equal to zero then the stock of this resource is full. Incrementing a counter means using one unit of the resource and resetting a counter means the full replenishment of the stock.

*This work has been partially supported by the EU CREDO project.

We define the language accepted by an R-automaton relative to a natural number D as the set of words allowing an accepting run of the automaton such that no counter value exceeds D in any state along the run. We study the problem of whether there is a number D such that the corresponding language is universal. This problem corresponds to the fact that with stock size D , the system can exhibit all the behaviors without running out of resources. We show that this problem is decidable in 2-EXPSpace. We extend this result to show decidability of the limitedness problem, i.e., to decide whether there is a natural number D such that all the accepted words can also be accepted with the counter value smaller than D . We also show the decidability of the non-emptiness problem. As a second technical contribution, we extend the decidability of the universality problem to R-automata with Büchi acceptance conditions.

To prove decidability of the universality problem, we adopt the technique from [Sim94] and extend it to our setting. We reformulate the problem in the language of finite monoids and solve it using the factorization forest theorem [Sim90]. In [Sim94], this theorem is used for solving the limitedness problem for distance automata. Distance automata are a subclass of R-automata with only one counter which is never reset. In contrast to this model, we handle several counters and resets. This extension cannot be encoded into the distance automata.

The decision algorithm deals with abstractions of collections of runs in order to find and analyze the loops created by these collections. The main step in the correctness proof is to show that each collection of runs along the same word can be split (factorized) into short repeated loops, possibly nested. Having such a factorization, one can analyze all the loops to check that none of the counters is only increased without being reset along them. If none of the counters is increased without being reset then we can bound the counter values by a constant derived from the length of the loops. Since the length of the loops is bounded by a constant derived from the automaton, all words can be accepted by a run with bounded counters. Otherwise, we show that there is a $+$ -free regular expression such that for any bound there is a word obtained by pumping this regular expression which does not belong to the language. Therefore, the language cannot be universal for any D .

Related work.

The concept of distance automata and the limitedness problem were introduced by Hashiguchi [Has82]. Different proofs of the decidability of the limitedness problem are reported in [Has90, Leu91, Sim94]. The last of these results [Sim94] is based on the factorization forest theorem [Sim90, Col07]. The model of R-automata, which we consider in this paper, extends that of distance automata by introducing resets and by allowing several counters. Furthermore, all the works mentioned above only consider the limitedness problem on finite words, while we here extend the decidability result of the universality problem to the case of infinite words. Distance automata were extended in [Kir05] with additional counters which can be reset following a hierarchical discipline

resembling parity acceptance conditions. R-automata relax this discipline and allow the counters to be reset arbitrarily. Universality of a similar type of automata for tree languages is studied in [CL08]. A model with counters which can be incremented and reset in the same way as in R-automata, called B-automata, is presented in [BC06]. B-automata accept infinite words such that the counters are bounded along an infinite accepting computation. Decidability of our problems can be obtained using the results from [BC06]. However, this would require complementation of a B-automaton which results in a non-elementary blowup of the automaton state space.

The fact that R-automata can have several counters which can be reset allows, for instance, to capture the abstractions of the sampled semantics of timed automata [KP05, AKY07]. A sampled semantics given by a sampling rate $\varepsilon = 1/f$ for some positive integer f allows time to pass only in steps equal to multiples of ε . The number of different clock valuations within one clock region (a bounded set of valuations) corresponds to a resource. It is finite for any ε while infinite in the standard (dense time) semantics of timed automata. Timed automata can generate runs along which clocks are forced to take different values from the same clock region (an increment of a counter), take exactly the same value (a counter is left unchanged), or forget about the previously taken values (a counter reset).

2 Preliminaries

First, we introduce the model of R-automata and its unparameterized semantics. Then, we introduce the parameterized semantics, the languages accepted by the automaton, and the decision problems.

R-automata. R-automata are finite state machines extended with counters. A transition may increase the value of a counter, leave it unchanged, or reset it back to zero. The automaton on its own does not have the capability of testing the values of the counters. However, the semantics of these automata is parameterized by a natural number D which defines an upper bound on counter values which may appear along the computations of the automaton. Let \mathbb{N} denote the set of non-negative integers.

An *R-automaton* with n counters is a 5-tuple $A = \langle S, \Sigma, \Delta, s_0, F \rangle$ where

- S is a finite set of states,
- Σ is a finite alphabet,
- $\Delta \subseteq S \times \Sigma \times \{0, 1, r\}^n \times S$ is a transition relation,
- $s_0 \in S$ is an initial state, and
- $F \subseteq S$ is a set of final states.

Transitions are labeled (together with a letter) by an effect on the counters. The symbol 0 corresponds to leaving the counter value unchanged, the symbol 1 represents an increment, and the symbol r represents a reset. We use t, t_1, \dots to denote elements of $\{0, 1, r\}^n$ which we call *effects*. A *path* is a sequences of transitions $(s_1, a_1, t_1, s_2), (s_2, a_2, t_2, s_3), \dots, (s_m, a_m, t_m, s_{m+1})$, such

that $\forall 1 \leq i \leq m. (s_i, a_i, t_i, s_{i+1}) \in \Delta$. An example of an R-automaton is given in Figure II.1.

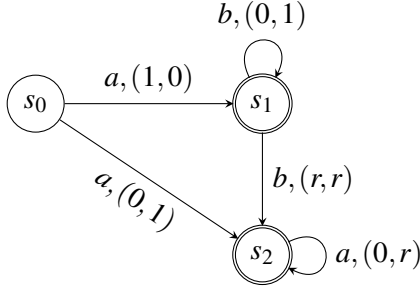


Figure II.1: An R-automaton with two counters.

Unparameterized semantics. We define an operation \oplus on the counter values as follows: for any $k \in \mathbb{N}$, $k \oplus 0 = k$, $k \oplus 1 = k + 1$, and $k \oplus r = 0$. We extend this operation to n -tuples by applying it componentwise. The operational semantics of an R-automaton $A = \langle S, \Sigma, \Delta, s_0, F \rangle$ is given by a labeled transition system (LTS) $\llbracket A \rrbracket = \langle \hat{S}, \Sigma, T, \hat{s}_0 \rangle$, where the set of states \hat{S} contains pairs $\langle s, (c_1, \dots, c_n) \rangle$, $s \in S, c_i \in \mathbb{N}$ for all $1 \leq i \leq n$, with the initial state $\hat{s}_0 = \langle s_0, (0, \dots, 0) \rangle$. The transition relation is defined by $(\langle s, (c_1, \dots, c_n) \rangle, a, \langle s', (c'_1, \dots, c'_n) \rangle) \in T$ if and only if $(s, a, t, s') \in \Delta$ and $(c'_1, \dots, c'_n) = (c_1, \dots, c_n) \oplus t$. We shall call the states of the LTS *configurations*.

We write $\langle s, (c_1, \dots, c_n) \rangle \xrightarrow{a} \langle s', (c'_1, \dots, c'_n) \rangle$ if $(\langle s, (c_1, \dots, c_n) \rangle, a, \langle s', (c'_1, \dots, c'_n) \rangle) \in T$. We extend this notation also for words, $\langle s, (c_1, \dots, c_n) \rangle \xrightarrow{w} \langle s', (c'_1, \dots, c'_n) \rangle$, where $w \in \Sigma^+$.

Paths in an LTS are called *runs* to distinguish them from paths in the underlying R-automaton. Observe that the LTS contains infinitely many states, but the counter values do not influence the computations, since they are not tested anywhere. In fact, for any R-automaton A , $\llbracket A \rrbracket$ is bisimilar to A considered as a finite automaton (without counters and effects). The LTS induced by the R-automaton from Figure II.1 is in Figure II.2.

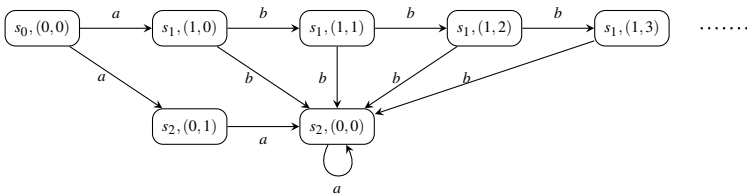


Figure II.2: The unparameterized semantics of the R-automaton in Figure II.1.

Parameterized Semantics. Next, we define the D -semantics of R-automata. We assume that the resources associated to the counters are not infinite and we can use them only for a bounded number of times before they

are replenished again. If a machine tries to use a resource which is already completely used up, it is blocked and cannot continue its computation.

For a given $D \in \mathbb{N}$, let \hat{S}_D be the set of configurations restricted to the configurations which do not contain a counter exceeding D , i.e., $\hat{S}_D = \{\langle s, (c_1, \dots, c_n) \rangle \mid \langle s, (c_1, \dots, c_n) \rangle \in \hat{S} \text{ and } (c_1, \dots, c_n) \leq (D, \dots, D)\}$ (\leq is applied componentwise). For an R-automaton A , the D -semantics of A , denoted by $\llbracket A \rrbracket_D$, is $\llbracket A \rrbracket$ restricted to \hat{S}_D . We write $\langle s, (c_1, \dots, c_n) \rangle \xrightarrow{a}_D \langle s', (c'_1, \dots, c'_n) \rangle$ to denote the transition relation of $\llbracket A \rrbracket_D$. We extend this notation for words, $\langle s, (c_1, \dots, c_n) \rangle \xrightarrow{w}_D \langle s', (c'_1, \dots, c'_n) \rangle$ where $w \in \Sigma^+$. The 2-semantics of the R-automaton from Figure II.1 is in Figure II.3.

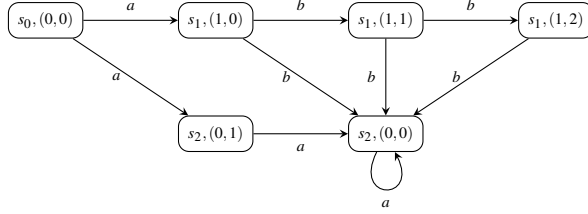


Figure II.3: The 2-semantics of the R-automaton in Figure II.1.

It is easy to see that for each $D_1 < D_2$, $\llbracket A \rrbracket_{D_2}$ simulates $\llbracket A \rrbracket_{D_1}$ and $\llbracket A \rrbracket$ simulates $\llbracket A \rrbracket_{D_2}$. Even stronger, for each $\hat{s} \in \hat{S}_{D_1}$, let $\hat{s}_{D_1}, \hat{s}_{D_2}, \hat{s}$ denote the configurations in $\llbracket A \rrbracket_{D_1}, \llbracket A \rrbracket_{D_2}, \llbracket A \rrbracket$, respectively. Then \hat{s}_{D_2} simulates \hat{s}_{D_1} and \hat{s} simulates \hat{s}_{D_2} .

We abuse the notation to avoid stating the counter values explicitly when it is not necessary. We define the reachability relations \longrightarrow and \longrightarrow_D over pairs of states and words as follows. For $s, s' \in S$ and $w \in \Sigma^+$, $s \xrightarrow{w} s'$ if and only if there is a path $(s, a_1, t_1, s_1), (s_1, a_2, t_2, s_2), \dots, (s_{|w|-1}, a_{|w|}, t_{|w|}, s')$ such that $w = a_1 \cdot a_2 \cdots a_{|w|}$. For each $D \in \mathbb{N}$, $s \xrightarrow{w}_D s'$ if also for all $1 \leq i \leq |w|$, $t_1 \oplus t_2 \oplus \cdots \oplus t_i \leq (D, \dots, D)$. It also holds that $s \xrightarrow{w}_D s'$ if and only if there is a run $\langle s, (0, \dots, 0) \rangle \xrightarrow{w}_D \langle s', (c_1, \dots, c_n) \rangle$.

Language. The (unparameterized or D -) language of an R-automaton is the set of words which can be read along the runs in the corresponding LTS ending in an accepting state (in a configuration whose first component is an accepting state). The *unparameterized language* accepted by an R-automaton A is $L(A) = \{w \mid s_0 \xrightarrow{w} s_f, s_f \in F\}$. For a given $D \in \mathbb{N}$, the D -language accepted by an R-automaton A is $L_D(A) = \{w \mid s_0 \xrightarrow{w}_D s_f, s_f \in F\}$. The unparameterized language of the R-automaton from Figure II.1 is ab^*a^* . The 2-language of this automaton is $a(\varepsilon + b + bb + bbb)a^*$.

Problem Definition. Now we can ask a question about language non-emptiness or universality of an R-automaton A parameterized by D , i.e., is there a natural number D such that $L_D(A) = \emptyset$ or $L_D(A) = \Sigma^*$. Figure II.4 shows an R-automaton A such that $L_2(A) = \Sigma^*$.

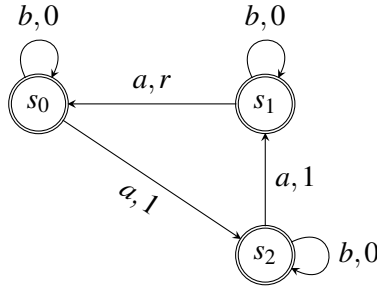


Figure II.4: A 2-universal R-automaton.

The language definitions and the questions can also be formulated for infinite words with Büchi acceptance conditions. The unparameterized ω -language of the automaton from Figure II.1 is $ab^\omega + ab^*a^\omega$. The 2- ω -language of this automaton is $a(\varepsilon + b + bb + bbb)a^\omega$.

3 Universality

The main result of the paper is the decidability of the universality problem for R-automata formulated in the following theorem.

Theorem 1 *For a given R-automaton A , the question whether there is $D \in \mathbb{N}$ such that $L_D(A) = \Sigma^*$ is decidable in 2-EXPSpace.*

First, we introduce and also formally define the necessary concepts (patterns, factorization, and reduction) together with an overview of the whole proof. Then we show the construction of the reduced factorization trees and state the correctness of this construction. Finally, we present an algorithm for deciding universality.

3.1 Concepts and Proof Overview

When an R-automaton A is not universal for all $D \in \mathbb{N}$ then there is an infinite set X of words such that for each $D \in \mathbb{N}$ there is $w_D \in X$ and $w_D \notin L_D(A)$. We say then that X is a counterexample. The main step of the proof is to show that there is an X which can be characterized by a $+$ -free regular expression. In fact, we show that X also satisfies a number of additional properties which enable us to decide for every such a $+$ -free regular expression, whether it corresponds to a counterexample or not. Another step of the proof is to show that we need to check only finitely many such $+$ -free regular expressions in order to decide whether there is a counterexample at all.

Patterns. The standard procedure for checking universality in the case of finite automata is subset construction. Whenever there are non-deterministic transitions $s \xrightarrow{a} s_1$ and $s \xrightarrow{a} s_2$ then we build a “summary” transition $\{s\} \xrightarrow{a} \{s_1, s_2\}$. This summary transition says that from the set of states $\{s\}$ we get to

the set of states $\{s_1, s_2\}$ after reading the letter a . In the case of R-automata, subset construction is in general not guaranteed to terminate since the values of the counters might grow unboundedly. To deal with this problem, we exploit the fact that the values of the counters do not influence the computations of the automaton. Therefore, we perform an abstraction which hides the actual values of the counters and considers only the effects along the transitions instead. The abstraction leads to a more complicated variant of summary transitions namely so called *patterns*.

We define a commutative, associative, and idempotent operation \circ on the set $\{0, 1, r\}$: $0 \circ 0 = 0$, $0 \circ 1 = 1$, $0 \circ r = r$, $1 \circ 1 = 1$, $1 \circ r = r$, and $r \circ r = r$. In fact, if we define an order $0 < 1 < r$ then \circ is the operation of taking the maximum. We extend this operation to effects, i.e., n -tuples, by applying it componentwise (this preserves all the properties of \circ). An effect obtained by adding several other effects through the application of the operator \circ summarizes the manner in which the counters are changed. More precisely, it describes whether a counter is reset or whether it is increased but not reset or whether it is only left untouched.

A pattern $\sigma : (S \times S) \longrightarrow 2^{\{0,1,r\}^n}$ is a function from pairs of automaton states to sets of effects. Let us denote patterns by $\sigma, \sigma_1, \sigma', \dots$. As an example, consider a pattern σ involving states s and s' and two counters. Let $\sigma(s, s) = \{(0, 0), (1, 1)\}$, $\sigma(s', s') = \{(1, 1), (1, 0)\}$, $\sigma(s, s') = \{(1, 1)\}$ and $\sigma(s', s) = \{(1, 1)\}$. This pattern is depicted in Figure II.5a.

Clearly, for a given R-automaton there are only finitely many patterns; let us denote this finite set of all patterns by \mathbb{P} . We define an operation \bullet on \mathbb{P} as follows. Let $(\sigma_1 \bullet \sigma_2)(s, s') = \{t \mid \exists s'', t_1, t_2. t_1 \in \sigma_1(s, s''), t_2 \in \sigma_2(s'', s'), t = t_1 \circ t_2\}$. Note, that \bullet is associative and it has a unit σ_e , where $\sigma_e(s, s') = \{(0, \dots, 0)\}$ if $s = s'$ and $\sigma_e(s, s') = \emptyset$ otherwise. Therefore, (\mathbb{P}, \bullet) is a finite monoid.

For each word we obtain a pattern by running the R-automaton along this word. Formally, let $\text{Run} : \Sigma^+ \longrightarrow \mathbb{P}$ be a homomorphism defined by $\text{Run}(a) = \sigma$, where $t \in \sigma(s, s')$ if and only if $(s, a, t, s') \in \Delta$.

Loops. In the case of finite automata, a set of states L and a word w constitute a loop in the subset construction if $L \xrightarrow{w} L$, i.e., starting from L and reading w , we end up in L again. The intuition behind the concept of a loop is that several iterations of the loop have the same effect as a single iteration. In our abstraction using patterns, loops are words w such that w yields the same pattern as w^2, w^3, \dots . We can skip the starting set of states, because the function Run starts implicitly from the whole set of states S (if there are no runs between some states then the corresponding set of effects is empty). More precisely, a word w is a loop if $\text{Run}(w)$ is an idempotent element of the pattern monoid. Two loops are identical if they produce the same pattern. Observe that the pattern in Figure II.5a is idempotent.

Factorization. We show that each word can be split into *short identical loops* repeated many times. The loops can possibly be nested, so that this split (*factorization*) defines a factorization tree. The idea is that since we have such a factorization for each word, it is sufficient to analyze only the (short) loops

and either find a run with bounded maximal value of the counters or use the loop structure to construct a counterexample regular expression.

On a higher level we can see a factorization of words as a function which for every word $w = a_1 a_2 \cdots a_l$ returns its factorization tree, i.e., a finite tree with branching degree at least 2 (except for the leaves) and with nodes labeled by subwords v of w such that the labeling function satisfies the following conditions:

- if a node labeled by v has children labeled by w_1, w_2, \dots, w_m then $v = w_1 \cdot w_2 \cdots w_m$,
- if $m \geq 3$ then $\sigma = \text{Run}(v) = \text{Run}(w_i)$ for all $1 \leq i \leq m$ and σ is idempotent,
- the leaves are labeled by a_1, a_2, \dots, a_l from left to right.

An example of such a tree is in Figure II.5b. It follows from the factorization forest theorem [Sim90, Col07] that there is such a (total) function which returns trees whose height is bounded by $3 \cdot |\mathbb{P}|$ where $|\mathbb{P}|$ is the size of the monoid.

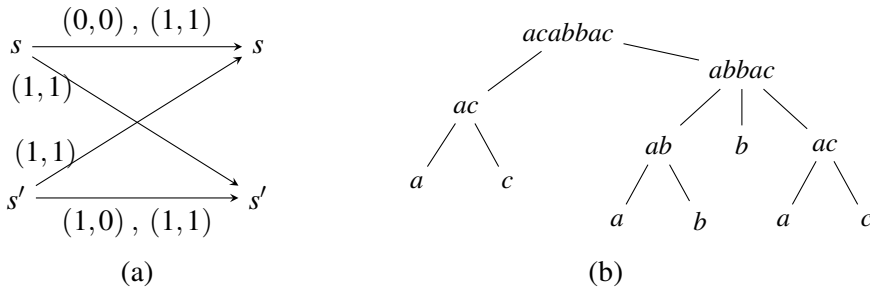


Figure II.5: A pattern involving two states and two counters (a) and a factorization tree (b). $\text{Run}(abbac) = \text{Run}(ab) = \text{Run}(b) = \text{Run}(ac)$ and it is idempotent.

We define the length of a loop as the length of the word (or a pattern sequence) provided that only the two longest iterations of the nested loops are counted. This concept is defined formally in Subsection 3.3. We say that the loops are short if there is a bound given by the automaton so that the length of all the loops is shorter than this bound. A consequence of the factorization forest theorem is that there is a factorization such that all loops are short.

Reduction. We have defined the loops so that the iterations of a loop have the same effect as the loop itself. Therefore, it is enough to analyze a single iteration to tell how the computations look when the loop is iterated an arbitrary number of times. By a *part* in an idempotent pattern σ , we mean an element (an effect) in the set $\sigma(s, s')$ for some states s and s' . We will distinguish between two types of parts, namely *bad* and *good* parts. A bad part corresponds only to runs along which the increase of some counter is at least as big as the number of the iterations of the loop. A part is *good* if there is a run with this effect along which the increase is bounded by the maximal increase induced by two iterations of the loop. Formally, we define a function *reduce* which for each pattern returns a pattern containing all good parts of the original pattern, but no bad parts. Then we illustrate it on a number of examples.

For a pattern σ , $\text{core}(\sigma)$ is defined as follows:

$$\text{core}(\sigma)(s, s') = \begin{cases} \sigma(s, s') \cap \{0, r\}^n & \text{if } s = s' \\ \emptyset & \text{otherwise} \end{cases}$$

Let $\text{reduce}(\sigma) = \sigma \bullet \text{core}(\sigma) \bullet \sigma$.

For an automaton with one state s , one counter, and a loop w with pattern σ , if $\sigma(s, s) = \{(1)\}$ then the whole pattern is bad, i.e., $\text{reduce}(\sigma)(s, s) = \emptyset$. Notice that any run over w^k increases the counter by k . On the other hand, if $\sigma(s, s) = \{(0)\}$ or $\sigma(s, s) = \{(r)\}$ then the whole pattern is good, i.e., $\text{reduce}(\sigma) = \sigma$.

With more complicated patterns we need a more careful analysis. Let us consider a loop w with pattern σ where $\sigma(s, s) = \{(0)\}$, $\sigma(s', s') = \{(1)\}$, $\sigma(s, s') = \{(1)\}$, and $\sigma(s', s) = \{(1)\}$. We will motivate why the part $(1) \in \sigma(s', s')$ is good. For any k , we can take the run over w^k which starts from s' , moves to s after the first iteration, stays in s for $k - 2$ iterations, and finally moves back to s' after the k^{th} iteration. Then, the effect of the run is (1) . Furthermore, the counter increase along the run is bounded by twice the maximal counter increase while reading w . In fact, using a similar reasoning, we can show that all parts of σ are good (which is consistent with the fact that $\text{reduce}(\sigma) = \sigma$).

As the last example, let us consider the pattern from Figure II.5a. First, we show that the part $(1, 0) \in \sigma(s', s')$ is bad. The only run over w^k with effect $(1, 0)$ is the one which comes back to s' after each iteration. However, this run increases the first counter by k . On the other hand, the part $(1, 1) \in \sigma(s', s')$ is good by a similar reasoning to the previous example. In fact, we can show that all other parts of the pattern are good (which is consistent with the value of $\text{reduce}(\sigma)$ in Figure II.6).

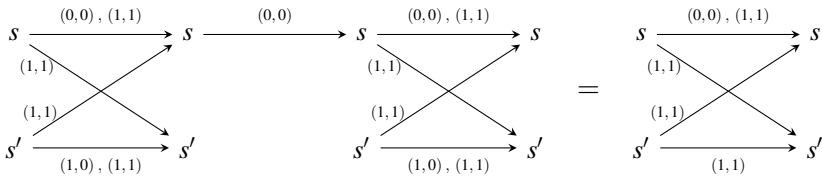


Figure II.6: $\sigma \bullet \text{core}(\sigma) \bullet \sigma = \text{reduce}(\sigma)$ where σ is the pattern from Figure II.5a

Reduced Factorization Trees. For a factorization of a word w , we need to check whether there is a run which goes through a good part in every loop. In order to do that, we enrich the tree structure, so that each node will now be labeled, in addition to a word, also by a pattern. The patterns are added by the following function: given an input sequence of patterns, the leaves are labeled by the elements of the sequence, nodes with branching degree 2 are labeled by the composition of the children labels, and we label each node with branching degree at least 3 by σ , where σ is the idempotent label of all its children. Now,

based on this labeling, we build a *reduced factorization tree* for w in several steps (formally described in Subsection 3.2).

We start with the sequence of patterns obtained by Run from the letters of the word. In each step, we take the resulting sequence from the previous step, build a factorization tree from it, and label it by patterns as described above. Then we take the lowest nodes such that they have at least 3 children and they are labeled by a pattern σ such that $\text{reduce}(\sigma) \neq \sigma$. We change the labels of these nodes to $\text{reduce}(\sigma)$. We pack the subtrees of these nodes into elements of the new sequence and we leave other elements of the sequence unmodified. This procedure eventually terminates and returns one tree with the following properties (the important invariant is shown in Lemma 2):

- if a node labeled by σ has two children labeled by σ_1, σ_2 then $\sigma = \sigma_1 \bullet \sigma_2$,
- if a node labeled by σ has m children labeled by $\sigma_1, \dots, \sigma_m$, $m \geq 3$, then $\sigma_i = \sigma_j$ for all $1 \leq i, j \leq m$, σ_1 is idempotent, and $\sigma = \text{reduce}(\sigma_1)$.

An example of a reduced factorization tree is in Figure II.7. We show that there is a factorization function such that the height of all reduced factorization trees produced by it is bounded by $3 \cdot |\mathbb{P}|^2$ (Lemma 5) using the factorization forest theorem and a property of the reduction function that if $\text{reduce}(\sigma) \neq \sigma$ then $\text{reduce}(\sigma) <_{\mathcal{J}} \sigma$, where $<_{\mathcal{J}}$ is the usual ordering of the \mathcal{J} -classes on \mathbb{P} , \mathcal{J} is a standard Green's relation; $\sigma \leq_{\mathcal{J}} \sigma'$ if and only if there are σ_1, σ_2 such that $\sigma = \sigma_1 \bullet \sigma' \bullet \sigma_2$; $\sigma <_{\mathcal{J}} \sigma'$ if and only if $\sigma \leq_{\mathcal{J}} \sigma'$ and $\sigma' \not\leq_{\mathcal{J}} \sigma$ (Lemma 3).

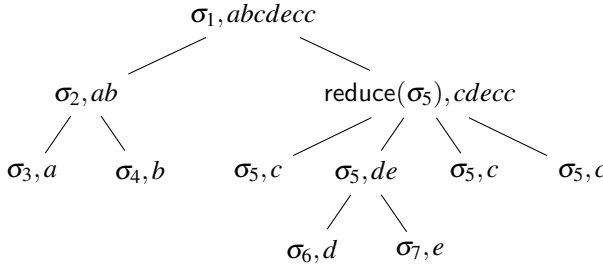


Figure II.7: An example reduced factorization tree. $\sigma_1 = \sigma_2 \bullet \text{reduce}(\sigma_5)$, $\sigma_2 = \sigma_3 \bullet \sigma_4$, and $\sigma_5 = \sigma_6 \bullet \sigma_7$. For all leaves labeled by $\hat{\sigma}$, $\hat{\sigma} = \text{Run}(\hat{\sigma})$.

Correctness. Let σ be the label of the root of a reduced factorization tree for a word w and let $\text{pump}(r, k)$ for a $+$ -free regular expression r and for a $k \in \mathbb{N}$ be the word obtained by repeating each r_1 , where r_1^* is a subexpression of r , k -times. Then

- if $\sigma(s_0, s_f) \neq \emptyset$ for some $s_f \in F$ then there is a run from s_0 to s_f over w in $8^{|\mathbb{P}|^2}$ -semantics,
- otherwise, there is a $+$ -free regular expression r such that for all D there is a k such that there is a counter which exceeds D along all runs from s_0 to s_f , $s_f \in F$, over $\text{pump}(r, k)$.

The previous items are formulated in Subsection 3.3, Lemma 7 and Lemma 9.

Relation to Simon's Approach. There are several important differences between the method presented in this paper and that of Simon [Sim94]. Our notion of pattern is a function to a set of effects, while in Simon's case it is a function to the set $\{0, 1, \omega\}$. Because of the resets and the fact that there are several counters, it is not possible to linearly order the effects. Thus, a collection of automaton runs can be abstracted into several incomparable effects. The sets are necessary in order to remember all of them. Furthermore, the different notion of pattern requires a new notion of reduction which does not remove loops labeled also by resets. We need to show then that application of this notion of reduction during the construction of the reduced factorization trees preserves the correctness.

3.2 Construction of the Reduced Factorization Tree

We define labeled finite trees to capture the looping structure of pattern sequences. Let Γ be a set of finite trees with two labeling functions Pat and Word , which for each node return a pattern and a word, respectively. We will abuse the notation and, for a tree T , we use $\text{Pat}(T)$ or $\text{Word}(T)$ to denote $\text{Pat}(N)$ or $\text{Word}(N)$, respectively, where N is the root of T . We also identify nodes with the subtrees in which they are roots. We can then say that a node T has children T_1, \dots, T_m and then use T_i 's as trees. For a tree T , we define its height $h(T)$ as $h(T) = 1$ if T is a leaf, $h(T) = 1 + \max\{h(T_1), \dots, h(T_m)\}$ if T_1, \dots, T_m are children of the root of T .

By Γ^+ we mean the set of nonempty sequences of elements of Γ . By $(\Gamma^+)^+$ we mean the set of nonempty sequences of elements of Γ^+ . Let us denote elements of Γ^+ by $\gamma, \gamma_1, \gamma', \dots$. For $\gamma \in \Gamma^+$, let $|\gamma|$ denote the length of γ .

Let $f : \Gamma^+ \rightarrow \mathbb{P}$ be a homomorphism with respect to \bullet defined by $f(T) = \text{Pat}(T)$. We call a function $d : \Gamma^+ \rightarrow (\Gamma^+)^+$ a *factorization function* if it satisfies the following conditions. If $d(\gamma) = (\gamma_1, \gamma_2, \dots, \gamma_m)$ then $\gamma = \gamma_1 \cdot \gamma_2 \cdots \gamma_m$, if $m = 1$ then $|\gamma| = 1$, and if $m \geq 3$ then $f(\gamma) = f(\gamma_i)$ for all $1 \leq i \leq m$ and $f(\gamma)$ is an idempotent element.

For a factorization function d we define two functions $\text{tree} : \Gamma^+ \rightarrow \Gamma$ and $\text{cons} : \Gamma^+ \rightarrow \Gamma^+$ inductively as follows. Let $\langle \sigma, w \rangle$ denote a tree which consists of only the root labeled by σ and w .

$$\text{tree}(\gamma) = \begin{cases} \gamma & \text{if } |\gamma| = 1, \\ \langle \sigma_1 \bullet \sigma_2, w_1 \cdot w_2 \rangle \text{ with children } \text{tree}(\gamma_1), \text{tree}(\gamma_2), & \text{if} \\ & d(\gamma) = (\gamma_1, \gamma_2), \sigma_i = \text{Pat}(\text{tree}(\gamma_i)), \text{ and} \\ & w_i = \text{Word}(\text{tree}(\gamma_i)) \text{ for } i \in \{1, 2\}, \\ \langle \text{reduce}(\sigma), w_1 \cdots w_m \rangle \text{ with children } \text{tree}(\gamma_1), \dots, \text{tree}(\gamma_m), & \\ & \text{if } m \geq 3, d(\gamma) = (\gamma_1, \gamma_2, \dots, \gamma_m), \\ & \sigma = \text{Pat}(\text{tree}(\gamma_1)), \text{ and } w_i = \text{Word}(\text{tree}(\gamma_i)) \\ & \text{for all } 1 \leq i \leq m. \end{cases}$$

The function tree builds a tree (resembling a factorization tree) from the sequence of trees according to the function d . The only difference from straightforwardly following the function d is that the labeling function Pat might be changed by the function reduce . Let us color the trees in the function cons either green or red during the inductive construction of a new sequence.

$$\text{cons}(\gamma) = \begin{cases} \gamma & \text{if } |\gamma| = 1. \text{ Mark } \gamma \text{ green.} \\ \text{cons}(\gamma_1) \cdot \text{cons}(\gamma_2) \cdots \text{cons}(\gamma_m) & \text{if } d(\gamma) = (\gamma_1, \gamma_2, \dots, \gamma_m) \text{ and either } m = 2 \text{ or} \\ & \text{there is } 1 \leq i \leq m \text{ such that } \text{cons}(\gamma_i) \text{ contains} \\ & \text{a red tree or } \text{reduce}(f(\gamma_i)) = f(\gamma_i). \\ \text{tree}(\gamma) & \text{if } d(\gamma) = (\gamma_1, \gamma_2, \dots, \gamma_m), m \geq 3, \text{ no } \text{cons}(\gamma_i) \\ & \text{contains a red tree and } \text{reduce}(f(\gamma_i)) \neq f(\gamma_i). \\ & \text{Mark the tree red.} \end{cases}$$

The function cons updates the sequence of trees trying to leave as much as possible untouched, but whenever Pat would be changed by the reduce function for the first time (on the lowest level), it packs the whole sequence into a single tree with changed Pat label of the root using the function tree.

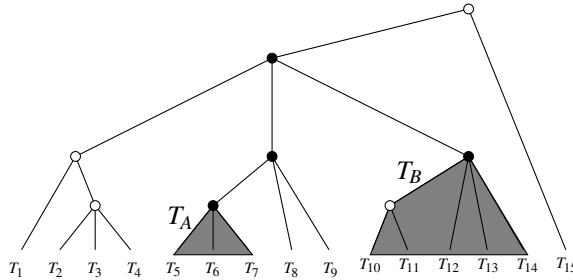


Figure II.8: Application of cons to $T_1 \cdots T_{15}$. The black nodes represent the nodes for which $\text{reduce}(\sigma) \neq \sigma$. The resulting sequence is $T_1 T_2 T_3 T_4 T_A T_8 T_9 T_B T_{15}$.

The important property of the construction is that for each tree in the new sequence it holds that whenever a node has more than two children, they are all labeled by identical idempotent patterns. Let us call a tree *balanced* if whenever a node T has children T_1, T_2, \dots, T_m , where $m \geq 3$, then $\text{Pat}(T_1) = \text{Pat}(T_2) = \dots = \text{Pat}(T_m)$, it is an idempotent element in \mathbb{P} , and $\text{Pat}(T) = \text{reduce}(\text{Pat}(T_1))$.

Lemma 2 For a $\gamma \in \Gamma^+$, if all trees in γ are balanced then all trees in $\text{cons}(\gamma)$ are balanced.

Proof. The only possibility where a new tree can occur in $\text{cons}(\gamma)$ is as a result of $\text{tree}(\gamma')$ for some γ' . The conditions on γ' are that $d(\gamma) = (\gamma_1, \dots, \gamma_m)$

and for all $1 \leq i \leq m$, $\text{cons}(\gamma_i)$ does not contain a red tree. Then we prove that $\text{Pat}(\text{tree}(\gamma)) = f(\gamma)$ for any $\gamma \in \Gamma^+$ such that $\text{cons}(\gamma)$ contains only green trees by induction on $h(\text{tree}(\gamma))$. If $h(\text{tree}(\gamma)) = 1$ then it follows directly from the definition of f . If $h(\text{tree}(\gamma)) > 1$ and $d(\gamma) = (\gamma_1, \gamma_2)$ then the claim follows from the induction hypothesis and the fact that f is a homomorphism. If $h(\text{tree}(\gamma)) > 1$ and $d(\gamma) = (\gamma_1, \dots, \gamma_m)$, $m \geq 3$, then the claim follows from the induction hypothesis and the fact that $\text{cons}(\gamma)$ contains only red trees, concretely, $\text{tree}(\gamma)$ is green, from which it follows that $\text{reduce}(f(\gamma_1)) = f(\gamma_1)$.

The fact that $\text{tree}(\gamma')$ is balanced follows directly from the previous property and the condition on the function d that $\text{Pat}(\gamma_i) = f(\gamma_i) = f(\gamma_i) = \text{Pat}(\gamma_i)$ for all $1 \leq i \leq m$. ■

Now we show how to get a sequence of trees from runs of the automaton. Let $\text{treeRun} : \Sigma^+ \rightarrow \Gamma^+$ be a homomorphism with respect to the word composition defined by $\text{treeRun}(a) = \langle \text{Run}(a), a \rangle$.

Assume that there is a factorization function d fixed. Let for a word $w \in \Sigma^+$, γ_w be defined as $\text{cons}^n(\text{treeRun}(w))$, where $n \in \mathbb{N}$ is the least such that $\text{cons}^n(\text{treeRun}(w)) = \text{cons}^{n+1}(\text{treeRun}(w))$. Note that γ_w is always defined, because for all $\gamma \in \Gamma^+$, $|\text{cons}(\gamma)| \leq |\gamma|$ and if $|\text{cons}(\gamma)| = |\gamma|$ then $\text{cons}(\gamma) = \gamma$. Let $T_w = \text{tree}(\gamma_w)$. We call T_w the *reduced factorization tree* of w . From Lemma 2 it follows that T_w is balanced (note that if $\text{cons}^n(\gamma) = \text{cons}^{n+1}(\gamma)$ then $\text{cons}^n(\gamma)$ contains only green trees).

Remark. Notice that we do not explicitly mention the factorization function d in the definition of a reduced factorization tree T_w constructed by d from a word w . It is always clear from the context which factorization function we mean.

To prove that the height of the reduced factorization trees is bounded for a given automaton, we need to show a technical property of the reduction function, namely that reduction strictly reduces the \mathcal{J} level of the pattern (\mathcal{J} is a standard Green's relation; $\sigma \leq_{\mathcal{J}} \sigma'$ if and only if there are σ_1, σ_2 such that $\sigma = \sigma_1 \bullet \sigma' \bullet \sigma_2$; $\sigma <_{\mathcal{J}} \sigma'$ if and only if $\sigma \leq_{\mathcal{J}} \sigma'$ and $\sigma' \not\leq_{\mathcal{J}} \sigma$).

Lemma 3 *For any idempotent pattern σ , either $\text{reduce}(\sigma) = \sigma$ or $\text{reduce}(\sigma) <_{\mathcal{J}} \sigma$.*

Proof. From the idempotence of σ it follows that $\text{reduce}(\sigma) = \sigma \bullet \text{reduce}(\sigma) \bullet \sigma$. This property is sufficient for the proof of Lemma 3 from [Sim94] which applies to our case. This proof uses Green's relations.

We present also an alternative proof without using Green's relations here.

First we show that if $\text{reduce}(\sigma) \neq \sigma$ then there are t and s such that $t \in \sigma(s, s)$ but $t \notin \text{reduce}(\sigma)(s, s)$. Assume that it is not the case. Because σ is idempotent and the function reduce does not add anything to the pattern, there are s, s', t such that $t \in \sigma(s, s')$, $t \notin \text{reduce}(\sigma)(s, s')$. Because σ is idempotent, there are s'', t_1, t_2, t_3 such that $t_1 \in \sigma(s, s'')$, $t_2 \in \sigma(s'', s'')$, $t_3 \in \sigma(s'', s'')$, $t = t_1 \circ t_2 \circ t_3$. From the assumption, $t_2 \in \text{reduce}(\sigma)(s'', s'')$, i.e., there are \hat{s}, t', t'', t''' such that $t' \in \sigma(s'', \hat{s})$, $t'' \in \text{core}(\sigma)(\hat{s}, \hat{s})$, $t''' \in \sigma(\hat{s}, s'')$, $t_2 = t' \circ t'' \circ t'''$. But because σ

is idempotent, $t_1 \circ t' \in \sigma(s, \hat{s})$ and $t''' \circ t_3 \in \sigma(\hat{s}, s')$, so $t \in \text{reduce}(\sigma)(s, s')$, which is a contradiction with the assumption.

Let us say that s and s' are merged by t in σ if $t \in \sigma(s, s), t \in \sigma(s', s'), t \in \sigma(s, s'), t \in \sigma(s', s)$. We write it $(s, t) \sim_m (s', t)$. In fact, for an idempotent pattern σ , the relation \sim_m is an equivalence relation on the set of pairs (s, t) . Note that if s, s' are merged by t in σ and $t \notin \text{reduce}(\sigma)(s, s)$ then $t \notin \text{reduce}(\sigma)(s', s')$. Therefore, the number of \sim_m equivalence classes of $\text{reduce}(\sigma)$ is strictly smaller than that of σ (unless they are equal).

Let $0 < 1 < r$. Let $t = (b_1, \dots, b_n) < t' = (b'_1, \dots, b'_n)$ if $b_i < b'_i$ for all $1 \leq i \leq n$. The set of effects together with this order is a finite lattice. Let $\downarrow t$ denote a principal ideal in this lattice generated by t . We try to construct σ_1, σ_2 so that $\sigma' = \sigma$, where $\sigma' = \sigma_1 \bullet \text{reduce}(\sigma) \bullet \sigma_2$, and we show that if we do not want to fail then $\text{reduce}(\sigma) = \sigma$.

Let us say that s, t where $t \in \sigma'(s, s)$ goes through s', t' if there are t_1, t_2, t_3, t_4, t_5 such that $t_1 \in \sigma_1(s, s_1), t_2 \in \sigma(s_1, s'), t_3 \in \text{core}(\sigma)(s', s'), t_4 \in \sigma(s', s_2), t_5 \in \sigma_2(s_2, s), t_3 < t'$, and $t' \in \sigma(s', s')$. The main idea of the rest of this proof is that to be able to construct i different equivalence classes wrt. \sim_m , we need i different equivalence classes in $\text{reduce}(\sigma)$. We will be interested only in the effects on the loops, i.e., only in $t \in \sigma'(s, s')$ where $s = s'$.

Note that if σ' is idempotent (and we want this, because σ is idempotent) then if s_1, t_1, s_2, t_2 go through s_3, t', s_4, t' , respectively, and $(s_3, t') \sim_m (s_4, t')$ in σ then $(s_1, t_1 \vee t_2) \sim_m (s_2, t_1 \vee t_2)$ in σ' . This follows from the idempotency of σ' and the definition of the relation *merged*; the reasoning is similar to the one in the first paragraph of this proof.

We show by induction on the size of $\downarrow t$ that if $t \in \sigma(s, s)$ for some s then we need as many equivalence classes which contain a $t' \in \downarrow t$ in their second component in $\text{reduce}(\sigma)$ as in σ to not to introduce any $t \in \sigma(s, s')$ such that $t \notin \sigma(s, s')$. The basic step is clear from the previous paragraph. For the induction step, if s, t goes through some s', t' such that $t' < t$ then $t \in \text{reduce}(\sigma)(s', s')$ must hold and thus it also goes through s', t . Also, each s, t, s', t which are not merged in σ have to go through s_1, t, s_2, t which are not merged in σ . Therefore, there are needed as many equivalence classes which contain t in their second component as there are in σ . ■

We state the factorization forest theorem. It was formulated and proved by Simon [Sim90], the best known bound is shown in [Col07].

Theorem 4 (Factorization Forest Theorem) *For a finite monoid \mathbb{P} and a homomorphism $f : \Gamma^+ \rightarrow \mathbb{P}$, there is a factorization function d such that for all $\gamma \in \Gamma^+, h(\text{tree}(\gamma)) \leq 3 \cdot |\mathbb{P}|$.*

We show that for each R-automaton there is a factorization function such that for any w the height of the tree T_w is bounded by a constant computed from the parameters of the automaton.

Lemma 5 *Given an R-automaton A , there is a factorization function d such that for all words $w \in \Sigma^+, h(T_w) \leq 3 \cdot |\mathbb{P}|^2$.*

Proof. Let us first define the nesting depth function $\text{nd} : \Gamma^+ \rightarrow \mathbb{N}$ by

$$\text{nd}(\gamma) = \begin{cases} 1 & \text{if } \gamma = \langle \sigma, a \rangle \\ 1 + \text{nd}(\gamma') & \text{if } |\gamma| = 1, \\ & \gamma \neq \langle \sigma, a \rangle, \\ & \gamma = \text{tree}(\gamma') \\ \max\{\text{nd}(T_i), \dots, \text{nd}(T_k)\} & \text{if } \gamma = T_1 \cdots T_k \end{cases}$$

Note that for any $w \in \Sigma^+$ and for any tree in γ_w , either the tree consists of only a root (it is equal to $\langle \sigma, a \rangle$ for some σ and a) or it has been obtained as $\text{tree}(\gamma')$ for some $\gamma' \in \Gamma^+$. Note also, that for each such tree, there is exactly one such γ' (for a fixed d). Therefore, the nesting depth function nd is well-defined for all γ_w .

From Lemma 3 it follows that whenever nd is applied to a γ such that $|\gamma| = 1$, $\gamma \neq \langle \sigma, a \rangle$, $\gamma = \text{tree}(\gamma')$, $\gamma' = T_1 \cdots T_k$ then for all $1 \leq i \leq k$, $\text{Pat}(\gamma) <_{\mathcal{J}} \text{Pat}(T_i)$. Thus, for any $w \in \Sigma$, $\text{nd}(\gamma_w) \leq |\mathbb{P}|$.

From Theorem 4, we know that there is d such that $h(\text{tree}(\gamma)) \leq \max\{h(T_1), \dots, h(T_k)\} + 3 \cdot |\mathbb{P}|$ for all sequences $\gamma = T_1 \cdots T_k$. Therefore, $h(T_w) = h(\text{tree}(\gamma_w))$, $h(\text{tree}(\gamma_w)) \leq 3 \cdot |\mathbb{P}| \cdot \text{nd}(\gamma_w) \leq 3 \cdot |\mathbb{P}|^2$ for this d . ■

3.3 Correctness

To formulate the first correctness lemma, we define the following concept of a length function $l : \Gamma \rightarrow \mathbb{N}$ inductively by

$$l(T) = \begin{cases} 1 & \text{if } T \text{ is a leaf} \\ l(T_1) + l(T_2) & \text{if } T \text{ has two children } T_1, T_2 \\ 2 \cdot \max\{l(T_1), \dots, l(T_m)\} & \text{if } T \text{ has children } T_1, \dots, T_m, m \geq 3 \end{cases}$$

By induction on $h(T_w)$ and using the bound derived in Lemma 5, one can show the following claim.

Lemma 6 *Given an R-automaton A, there is a factorization function d such that for all words $w \in \Sigma^+$, $l(T_w) \leq 8^{|\mathbb{P}|^2}$.*

Proof. By induction on $h(T_w)$ and using the bound derived in Lemma 5. ■

We say that $s \xrightarrow{w} s'$ or $s \xrightarrow{w}_D s'$ realizes t if there is a witnessing path $(s, a_1, t_1, s_1), (s_1, a_2, t_2, s_2), \dots, (s_{|w|-1}, a_{|w|}, t_{|w|}, s')$ such that $t = t_1 \circ t_2 \circ \dots \circ t_{|w|}$. If $s \xrightarrow{w}_D s'$ (or $s \xrightarrow{w} s'$) realizes $t = (b_1, \dots, b_n)$, the counter values along a run $\langle s, (c_1, \dots, c_n) \rangle \xrightarrow{w} \langle s', (c'_1, \dots, c'_n) \rangle$ produced by this path satisfy the following conditions:

- if $b_i = 0$ then $c_i = c'_i$ for all states $\langle s'', (c''_1, \dots, c''_n) \rangle$ along the run,
- if $b_i = r$ then $c''_i = 0$ (since it is reset) in some state $\langle s'', (c''_1, \dots, c''_n) \rangle$ along the run, and

- if $b_i = 1$ then $c_i < c'_i$ (and it is not reset along the run).

Let us define $\text{Run}_D(w)$ to be the pattern obtained by running the automaton over w in the D -semantics. Formally, $\text{Run}_D(w)(s, s')$ contains t if and only if $s \xrightarrow{w}_D s'$ realizes t . Note that the function Run_D is not a homomorphism with respect to the word composition. We also define a relation \sqsubseteq on patterns by $\sigma \sqsubseteq \sigma'$ if and only if for all s, s' , $\sigma(s, s') \subseteq \sigma'(s, s')$.

From Lemma 6 we show that there is a factorization function such that for every w , $\text{Pat}(T_w)$ corresponds to the runs of the R-automaton which can be performed in the D -semantics for any big enough D . This is formulated in the following lemma.

Lemma 7 *Given an R-automaton, there is a factorization function such that for all $w \in \Sigma^+$ and for all $D \in \mathbb{N}, D \geq 8^{|\mathbb{P}|^2}$, $\text{Pat}(T_w) \sqsubseteq \text{Run}_D(w)$.*

Proof. Let us fix a factorization function d satisfying Lemma 6. We show this lemma by proving the following claim by induction on $h(T_w)$. For any $w \in \Sigma^+$, if $t \in \text{Pat}(T_w)(s, s')$ then $s \xrightarrow{w}_D s'$ realizing t for $D = l(T_w)$. From Lemma 6 we have that such a run exists also in any D -semantics for $D \geq 8^{|\mathbb{P}|^2}$.

The basic step follows directly from the definition of the function treeRun .

Assume that the tree has the root $\langle \sigma_1 \bullet \sigma_2, w_1 \cdot w_2 \rangle$ with children T_{w_1} and T_{w_2} (note that for each subtree T , $T = T_{\text{Word}(T)}$), where $\sigma_1 = \text{Pat}(T_{w_1})$, $\sigma_2 = \text{Pat}(T_{w_2})$. Then there are s'', t_1, t_2 such that $t_1 \in \sigma_1(s, s'')$, $t_2 \in \sigma_2(s'', s')$, and $t = t_1 \circ t_2$. From the induction hypothesis, $s \xrightarrow{w_1}_{D_1} s''$ realizes t_1 and $s'' \xrightarrow{w_2}_{D_2} s'$ realizes t_2 , where $D_1 = l(T_{w_1}), D_2 = l(T_{w_2})$. Clearly, if we concatenate any two paths given by these relations, we get $s \xrightarrow{w}_{D_1+D_2} s'$ realizing $t_1 \circ t_2$. From the definition of the length function, $l(T_w) = l(T_{w_1}) + l(T_{w_2}) = D_1 + D_2$.

Assume that the tree has the root $\langle \text{reduce}(\sigma), w_1 \cdots w_m \rangle$ with children T_{w_1}, \dots, T_{w_m} , where $m \geq 3, \sigma = \text{Pat}(T_{w_1})$. Then there are s'', t_1, t_2, t_3 such that $t_1 \in \sigma(s, s'')$, $t_2 \in \sigma(s'', s')$, $t_3 \in \sigma(s'', s')$, $t = t_1 \circ t_2 \circ t_3$, and $t_2 \in \{0, r\}^n$ (this follows directly from the definition of the function reduce). Since $\text{Pat}(T_{w_i}) = \sigma$ for all $1 \leq i \leq m$ ((which we have from Lemma 2) then from the induction hypothesis $s \xrightarrow{w_1}_{l(T_{w_1})} s''$ realizes t_1 , $s'' \xrightarrow{w_i}_{l(T_{w_i})} s''$ realizes t_2 for all $2 \leq i \leq m-1$, and $s'' \xrightarrow{w_m}_{l(T_{w_m})} s'$ realizes t_3 .

Let us analyze the length of the concatenation of the paths given by these relations. For each counter, if its corresponding effect in t_2 is 0 then the bound on this counter during the whole path is $l(T_{w_1}) + l(T_{w_m})$, because it is left unchanged during the path part over $w_2 \cdot w_3 \dots w_{m-1}$. If the corresponding effect in t_2 of the counter is r then the counter is reset at least once in each path part over w_2, w_3, \dots, w_{m-1} . Therefore, it is bounded by the maximal length between two resets, which is bounded by $\max\{l(T_{w_1}) + l(T_{w_2}), l(T_{w_2}) + l(T_{w_3}), \dots, l(T_{w_{m-1}}) + l(T_{w_m})\}$. Then, $s \xrightarrow{w}_D s'$ realizes t , where $D = 2 \cdot \max\{l(T_{w_1}), \dots, l(T_{w_m})\}$. ■

Of particular interest are runs starting in the initial state.

Corollary 8 *Given an R -automaton A , there is a factorization function such that for all words w , if $\text{Pat}(T_w)(s_0, s) \neq \emptyset$ then there is a run $\langle s_0, (0, \dots, 0) \rangle \xrightarrow{w}_D \langle s, (c_1, \dots, c_n) \rangle$ where $D = l(T_w)$.*

It remains to show that if the relation between the patterns in the previous lemma is strict then there is a word for each D which is a witness for the strictness, i.e., the runs over this word in the D -semantics generate a smaller pattern than over the original word. These witness words are generated from a $+$ -free regular expression r by pumping r_1 for all subexpressions r_1^* of r . Let us define a function re which for a reduced factorization tree returns a $+$ -free regular expression inductively by

$$\text{re}(T) = \begin{cases} \text{Word}(T) & \text{if } T \text{ is a leaf} \\ \text{re}(T_1) \cdot \text{re}(T_2) & \text{if } T \text{ has two children } T_1, T_2 \\ (\text{re}(T_1))^* & \text{if } T \text{ has children } T_1, T_2, \dots, T_m, m \geq 3 \end{cases}$$

For a $+$ -free regular expression r and a natural number $k > 0$, let the function $\text{pump}(r, k)$ be defined inductively as follows: $\text{pump}(a, k) = a$, $\text{pump}(r_1 \cdot r_2, k) = \text{pump}(r_1, k) \cdot \text{pump}(r_2, k)$, and $\text{pump}(r^*, k) = \text{pump}(r, k)^k$.

For example, $\text{pump}(a(bc^*d)^*aa^*, 2) = abccdbccdaaa$.

Lemma 9 *Given an R -automaton and a factorization function, for all $w \in \Sigma^+$ and all $D \in \mathbb{N}$ there is a $k \in \mathbb{N}$ such that $\text{Run}_D(\text{pump}(\text{re}(T_w), k)) \subseteq \text{Pat}(T_w)$.*

Proof. We show this lemma by proving the following claim by induction on $h(T_w)$. For all $D \in \mathbb{N}$ there is $k \in \mathbb{N}$ such that for $v = \text{pump}(\text{re}(T_w), k)$, if $s \xrightarrow{v}_D s''$ realizes t then $t \in \text{Pat}(T_w)(s, s')$ (note that this holds also for all $k' > k$).

The basic step follows directly from the definition of the function treeRun (with any k).

Assume that the tree has the root $\langle \sigma_1 \bullet \sigma_2, w_1 \cdot w_2 \rangle$ with children T_{w_1} and T_{w_2} , where $\sigma_1 = \text{Pat}(T_{w_1})$, $\sigma_2 = \text{Pat}(T_{w_2})$. Let k_1, k_2 be the constants from the induction hypothesis applied to T_{w_1} and T_{w_2} . Let $k = \max\{k_1, k_2\}$. Let us denote $v_1 = \text{pump}(\text{re}(T_{w_1}), k)$, $v_2 = \text{pump}(\text{re}(T_{w_2}), k)$, $v = v_1 \cdot v_2 = \text{pump}(\text{re}(T_w), k)$. Assume that $s \xrightarrow{v}_D s''$ realizes t . Then there must be an s'' such that $s \xrightarrow{v_1}_D s''$, $s'' \xrightarrow{v_2}_D s'$ realize t_1, t_2 , respectively, such that $t = t_1 \circ t_2$. From the induction hypothesis, $t_1 \in \text{Pat}(T_{w_1})(s, s'')$ and $t_2 \in \text{Pat}(T_{w_2})(s'', s')$. Because $\text{Pat}(T_w) = \sigma_1 \bullet \sigma_2 = \text{Pat}(T_{w_1}) \bullet \text{Pat}(T_{w_2})$, we have that $t = t_1 \circ t_2 \in \text{Pat}(T_w)(s, s')$.

Assume that the tree has the root $\langle \text{reduce}(\sigma), w_1 \cdots w_m \rangle$ with children T_{w_1}, \dots, T_{w_m} , where $m \geq 3$, $\sigma = \text{Pat}(T_{w_1})$. Let k_1 be the constant from the induction hypothesis applied to T_{w_1} and $k_2 = (D+1)^n \cdot |S|$. Let $k = \max\{k_1, k_2\}$. Let us denote $v_1 = \text{pump}(\text{re}(T_{w_1}), k)$, $v = v_1^k = \text{pump}(\text{re}(T_w), k)$.

Assume that $s \xrightarrow{v}_D s'$ realizes t . Then there must be a sequence of states s_i for $1 \leq i \leq k+1$ such that $s_i \xrightarrow{v_1}_D s_{i+1}$ realizes t_i , $s_1 = s, s_{k+1} = s'$, and $t = t_1 \circ t_2 \circ \dots \circ t_k$. First, we show by contradiction that there are indices i, j such that $i < j, s_i = s_j$ and $t_i \circ \dots \circ t_{j-1} \in \{0, r\}^n$. Let us assume that for all $i < j$

such that $s_i = s_j$, $t_i \circ \dots \circ t_{j-1} \notin \{0, r\}^n$. Let us pick an \hat{s} such that $G = |\{i | s_i = \hat{s}, 1 \leq i \leq k+1\}|$ is maximal. From the choice of k we have that $G > D^n$. We show that there is a counter exceeding D along all paths witnessing $s \xrightarrow{v} s'$ realizing t . We know from our assumption ($t_i \circ \dots \circ t_{j-1} \notin \{0, r\}^n$) and from the definition of realizing that for all i, j such that $s_i = s_j = \hat{s}$, the counter values in any run over v cannot be identical in s_i and s_j . There are D^n different configurations with all counters smaller than or equal to D . Since $G > D$, some counter has to exceed D . This contradicts that $s \xrightarrow{v} s'$ realizes t .

From the induction hypothesis we have that for all $1 \leq i \leq k$, $t_i \in \text{Pat}(T_{w_1})$. Let i and j satisfy the condition from the previous paragraph, i.e., $i < j, s_i = s_j$ and $t_i \circ \dots \circ t_{j-1} \in \{0, r\}^n$. Because $\text{Pat}(T_{w_1})$ is idempotent (follows from Lemma 2), we have that $t_i \circ \dots \circ t_{j-1} \in \text{Pat}(T_{w_1})(s_i, s_j)$ and thus $t_i \circ \dots \circ t_{j-1} \in \text{core}(\text{Pat}(T_{w_1}))(s_i, s_j)$. Also, $t_1 \circ \dots \circ t_{i-1} \in \text{Pat}(T_{w_1})(s, s_i)$ and $t_j \circ \dots \circ t_k \in \text{Pat}(T_{w_1})(s_j, s')$. From the definition of the function `reduce`, we can conclude that $t \in \text{reduce}(\text{Pat}(T_{w_1}))(s, s')$. ■

A special case are runs starting from the initial state.

Corollary 10 *Given an R-automaton, for any $w \in \Sigma^+$, if $\text{Pat}(T_w)(s_0, s) = \emptyset$ then $\forall D \exists k$ such that there is no run $\langle s_0, (0, \dots, 0) \rangle \xrightarrow{v} \langle s, (c_1, \dots, c_n) \rangle$ where $v = \text{pump}(\text{re}(T_w), k)$.*

3.4 Algorithm

To check the universality of an R-automaton A , we have to check all patterns σ such that $\sigma = \text{Pat}(T_w)$ for some $w \in \Sigma^+$ and some factorization function. If there is a σ such that for all $s_f \in F$, $\sigma(s_0, s_f) = \emptyset$ then for all $D \in \mathbb{N}$, $L_D(A) \neq \Sigma^*$. This gives us the following algorithm. Recall that σ_e denotes the unit of (\mathbb{P}, \bullet) .

The algorithm uses a set of patterns P as the data structure. Given an R-automaton $A = \langle S, \Sigma, \Delta, s_0, F \rangle$ on the input, it answers 'YES' or 'NO'. The set P is initialized by $P = \{\sigma | \sigma = \text{Run}(a), a \in \Sigma\} \cup \{\sigma_e\}$.

While $|P|$ increases the algorithm performs the following operations:

- pick $\sigma_1, \sigma_2 \in P$ and add $\sigma_1 \bullet \sigma_2$ back to P .
- pick a $\sigma \in P$ such that σ is idempotent and add `reduce`(σ) back to P .

If there is $\sigma \in P$ such that for all $s_f \in F$, $\sigma(s_0, s_f) = \emptyset$, answer 'NO', otherwise, answer 'YES'.

Before we prove the correctness of the algorithm, we show that each pattern obtained by the algorithm corresponds to some word and some factorization function.

Lemma 11 *For any $\sigma \in \mathbb{P}$ obtained by the algorithm there is a factorization function and a word w such that $\sigma = \text{Pat}(T_w)$.*

Proof. Consider the tree labeled by the patterns defined inductively as follows. The root is labeled by σ . If a node is labeled by σ' which was created (for the first time) by composing $\sigma_1 \bullet \sigma_2$ then this node has two children labeled by

σ_1 and σ_2 . If a node is labeled by σ' which was created (for the first time) by reducing σ_1 then this node has one child labeled by σ_1 . The leaf labels have been added in the initialization step. Clearly, if $\sigma_1 = \sigma_2$ are labels of two nodes in the tree then their subtrees are identical.

Now we define a partial function $w : \mathbb{P} \rightarrow \Sigma^+$ which for each pattern in the tree returns a word and if $\sigma_1 \neq \sigma_2$ then $w(\sigma_1) \neq w(\sigma_2)$. Such a labeling also defines a factorization function which for $w = w(\sigma)$ yields the tree T_w such that $\sigma = \text{Pat}(T_w)$.

We start from the leaves and move inductively up. During the whole construction, we maintain a counter c , which is initially set to $c = 1$. For each σ in a leaf, $w(\sigma) = a$ such that $\text{Run}(a) = \sigma$ (if there are several, we assume some ordering and pick the least one). If a node is labeled by σ' and it has two children labeled by σ_1 and σ_2 then $w(\sigma') = w(\sigma_1) \cdot w(\sigma_2)$. If a node is labeled by σ' and it has one child labeled by σ_1 then $w(\sigma') = (w(\sigma_1))^k$ such that $|\mathbb{P}|^c < |w(\sigma')| \leq 2 \cdot |\mathbb{P}|^c$ and we increment c .

For two different patterns such that at least one of them has a reduction in its subtree, the words have to have a different length. For two different patterns such that there is no reduction in their subtrees, the words have to be different because of the definition of Run and \bullet (and all such words are shorter than $|\mathbb{P}|$).

■

The correctness is stated in the following theorem.

Theorem 12 *The algorithm is correct and runs in 2-EXPSPACE.*

Proof. Clearly, the algorithm "checks" all possible σ 's such that there is a factorization function and a word w such that $\sigma = \text{Pat}(T_w)$. Also, for any σ obtained by the algorithm there is a factorization function and a word w such that $\sigma = \text{Pat}(T_w)$ (Lemma 11), with the exception of σ_e which corresponds to $w = \varepsilon$ (for which is the correctness clear).

If the algorithm obtains a σ such that $\sigma(s_0, s_f) = \emptyset$ for all $s_f \in F$ then let us fix a factorization function and a word w such that $\sigma = \text{Pat}(T_w)$. Let $r = \text{re}(T_w)$. From Corollary 10, for all D there is a k such that there is no accepting run over $\text{pump}(r, k)$ in D -semantics.

If for all patterns σ , $\sigma(s_0, s_f) \neq \emptyset$ for some $s_f \in F$ then we can fix a factorization function satisfying Lemma 6. For all words, there is an accepting run in $8^{|\mathbb{P}|^2}$ -semantics given by Corollary 8.

The complexity follows from the size of the monoid \mathbb{P} . The algorithm needs space $|\mathbb{P}|$ (the number of different patterns). The size of \mathbb{P} is $2^{(3^n) \cdot |S|^2}$ ($|S|^2$ different pairs of states, $2^{(3^n)}$ different sets of effects). Therefore, the algorithm needs double exponential space.

■

4 Limitedness

The presented method can be adapted to decide the limitedness problem for R-automata, i.e., given an R-automaton A , is there a $D \in \mathbb{N}$ such that $L(A) = L_D(A)$.

Theorem 13 *For a given R-automaton A , the limitedness problem is decidable in 2-EXPSPACE.*

To decide the limitedness problem of an R-automaton A , we need to adapt the basic concepts of the method. Effects are elements of the set $\{0, 1, r, \omega\}^n$. We extend \circ by defining $\omega \circ b = b \circ \omega = \omega$ for all $b \in \{0, 1, r, \omega\}$. Patterns are then functions $\sigma : (S \times S) \longrightarrow 2^{\{0,1,r,\omega\}^n}$. The definition of \bullet remains the same and patterns together with \bullet form a finite monoid.

For an effect t , let \hat{t} denote the result of replacing 1's in t by ω 's. The function core is modified as follows. For each pattern σ , $\hat{t} \in \text{core}(\sigma)(s, s')$ if and only if $t \in \sigma(s, s')$ and $s = s'$. For each σ , $\text{reduce}(\sigma) <_{\mathcal{J}} \sigma$, because $\text{reduce}(\sigma) = \sigma \bullet \text{core}(\sigma) \bullet \sigma$ (Lemma 3 in Appendix). This gives us the boundedness of the height of the reduced factorization trees constructed with the new reduction function.

It holds that $\text{Pat}(T_w)(s, s') \neq \emptyset$ if and only if $s \xrightarrow{w} s'$. Moreover, Lemma 7 and Lemma 9 hold if we restrict the resulting pattern $\text{Pat}(T_w)$ to $\{0, 1, r\}$ (for all s, s' , we consider only $\text{Pat}(T_w)(s, s') \cap \{0, 1, r\}^n$). Our proofs can be modified in a straightforward manner, since whenever an ω occurs in an effect it cannot be overwritten any time later.

The condition for concluding non-limitedness of the input R-automaton in the algorithm is changed to checking whether there is $\sigma \in P$ such that the following two conditions hold: (i) there is $s_f \in F$, $\sigma(s_0, s_f) \neq \emptyset$ and (ii) for all $s_f \in F$, $\sigma(s_0, s_f) \cap \{0, 1, r\}^n = \emptyset$.

5 Büchi Universality

The universality problem is also decidable for R-automata with Büchi acceptance conditions.

Theorem 14 *For a given R-automaton A , the question whether there is $D \in \mathbb{N}$ such that $L_D^\omega(A) = \Sigma^\omega$ is decidable in 2-EXPSPACE.*

To show this result, we need to extend patterns by accepting state information. A pattern is now a function $\sigma : S \times S \longrightarrow 2^{\{0,1\} \times \{0,1,r\}^n}$, where for s, s' and $\langle a, t \rangle \in \sigma(s, s')$, the value of a encodes whether there is a path from s to s' realizing t which meets an accepting state. For instance, $\sigma(s, s') = \{\langle 0, (0, r) \rangle, \langle 1, (1, 1) \rangle\}$ means that there are two different types of paths between s and s' : they either realize $(0, r)$ but do not visit an accepting state, or realize $(1, 1)$ and visit an accepting state. We define the composition \bullet by defining the composition on the accepting state: $0 \circ 0 = 0, 0 \circ 1 = 1 \circ 0 = 1 \circ 1 = 1$. The set of patterns (denote again \mathbb{P}) with \bullet is a finite monoid. We define the

function reduce in the same way as before, i.e., the accepting state information does not play any role there. Clearly, $\text{reduce}(\sigma) < \not\sim \sigma$, so the reduced factorization trees produced by reduce have bounded height. Lemma 7 and Lemma 9 also hold, because (non)visiting an accepting state does not influence the runs in the D -semantics.

This allows us to use the same algorithm as for the finite word universality problem, except for the condition for concluding non-universality. The condition is whether there are $\sigma_1, \sigma_2 \in P$ such that σ_2 is idempotent and for all s such that $\sigma_1(s_0, s) \neq \emptyset$ the following holds. If $\langle a, t \rangle \in \sigma_2(s, s)$ then either $a = 0$ or $t \notin \{0, r\}^n$.

Proof. Let us denote the new pattern function by Pat^B and the new function which extracts a pattern from the runs in the D -semantics by Run_D^B .

Let for an R-automaton, $C = 8^{|\mathbb{P}|^2}$ and for an ω -word w , $w = w_1 \cdot w_2 \cdot w_3 \cdots$ be a split of this word such that all w_i are finite. For each w_i we define a pattern σ_{w_i} which captures the effects of the corresponding fragments of all infinite runs over w in $2 \cdot C$ -semantics. The choice of $2 \cdot C$ is motivated by the reasons explained below. Let for all $1 \leq i$, σ_{w_i} be a pattern defined by $\langle a, t \rangle \in \sigma_{w_i}(s, s')$ if and only if there is an infinite run in $2 \cdot C$ -semantics $\langle s_0, (0, \dots, 0) \rangle \xrightarrow{w_1 \cdots w_{i-1}} 2 \cdot C \langle s, (c_1, \dots, c_n) \rangle \xrightarrow{w_i} 2 \cdot C \langle s', (c'_1, \dots, c'_n) \rangle \xrightarrow{w_{i+1} \cdots}$ such that the fragment $\langle s, (c_1, \dots, c_n) \rangle \xrightarrow{w_i} \langle s', (c'_1, \dots, c'_n) \rangle$ realizes t and $a = 1$ if and only if this fragment contains an accepting state.

Assume that for all D , the R-automaton is not Büchi universal in the D -semantics. Let w be a counterexample for $D = 2 \cdot |\mathbb{P}| \cdot C$, i.e., $w \notin L_D^\omega(A)$. Let us split $w = w_1 \cdot w_2 \cdot w_3 \cdots$ so that all w_i are finite and $\sigma_{w_i} = \sigma_{w_j}$ for all $2 \leq i, j$. Let us denote $\sigma_1 = \sigma_{w_1}$ and $\sigma_2 = \sigma_{w_2}$.

Let $l \in \mathbb{N}$ be such that σ_2^l is an idempotent ($l \leq |\mathbb{P}|$). For all s , $\sigma_2^l(s, s)$ does not contain $\langle 1, (b_1, \dots, b_n) \rangle$, where $b_i \in \{0, r\}$. Otherwise, i.e., if there was $s, t \in \{0, r\}^n$ such that $\langle 1, t \rangle \in \sigma_2^l(s, s)$, there would be an accepting infinite run over w in the D -semantics, which would contradict the fact that $w \notin L_D^\omega(A)$. This follows from the fact that all patterns were obtained in the $2 \cdot C$ -semantics and $l \leq |\mathbb{P}|$.

It is not necessary that $\text{Run}_C^B(w_2) \sqsubseteq \sigma_2$, because the set of starting states for Run_C^B is S . Even if we restrict the set of starting states to $L(\sigma_2)$, denoted $\text{Run}_C^B(w_2)'$, the relation $\text{Run}_C^B(w_2)' \sqsubseteq \sigma_2$ does not have to hold. This is because a fragment of a run over w_2 in $2 \cdot C$ -semantics could have started from a state with high counter values and Run_C^B starts from zeros. However, if we restrict the set of starting states to the states which are in $2 \cdot C$ -semantics reachable after reading w_1 with counter values smaller than C , denote $\hat{\text{Run}}_C^B(w_2)$, then $\hat{\text{Run}}_C^B(w_2) \sqsubseteq \sigma_2$ holds, because now Run_C^B starts from zeros and is limited by C , whereas σ_2 contains all runs which start from counter values smaller than C and they are limited by $2 \cdot C$.

From Lemma 7 we know that there is a factorization function such that $\sigma_3 = \text{Pat}^B(T_{w_1}) \sqsubseteq \text{Run}_C^B(w_1)$ and $\sigma_4 = \text{Pat}^B(T_{w_2}) \sqsubseteq \text{Run}_C^B(w_2)$. Let m be such that σ_4^m is idempotent. Note that $\sigma_4^m \sqsubseteq \sigma_2^l$. We know that if $\sigma_3 \bullet \sigma_4^m(s_0, s) \neq \emptyset$ then $\sigma_4^m(s, s)$ does not contain $\langle 1, (b_1, \dots, b_n) \rangle$, where $b_i \in \{0, r\}$. Therefore,

from Lemma 9 we know that for any factorization function it holds that for all D there is a k such that $\text{pump}(\text{re}(T_{w_1}), k) \cdot (\text{pump}(\text{re}(T_{w_2}), k))^\omega \notin L_D^\omega(A)$. ■

6 Non-emptiness

The language emptiness problems can be decided much easier. The following theorem follows easily from an observation that for an R-automaton A , $\llbracket A \rrbracket$ is bisimilar to A taken as a finite automaton (without counters).

Theorem 15 *For a given R-automaton A , the question whether there is $D \in \mathbb{N}$ such that $L_D(A) \neq \emptyset$ is decidable in NLOGSPACE.*

To show the Büchi case, one has to find an accepting loop which for each counter either contains an r or contains only 0's. This information can be computed using an abstraction of the reachability information.

Theorem 16 *For a given R-automaton A , the question whether there is $D \in \mathbb{N}$ such that $L_D^\omega(A) \neq \emptyset$ is decidable in PSPACE.*

A *reachability abstraction* C is a set of the elements from $\{0, 1, r\}^n$ (a set of n -tuples of 0's, 1's, and r 's). For a given R-automaton A and states s, s' of A , we use $C_{s, s'}$ to denote a reachability abstraction defined as follows: $t \in C_{s, s'}$ if and only if there is $w \in \Sigma^+$ such that $s \xrightarrow{w} s'$ realizes t .

In particular, if s' is not reachable from s then $C_{s, s'} = \emptyset$. If $t \in C_{s, s'}$ and $t \in \{0, r\}^n$ then there is a loop such that for a sufficiently big D the loop can be iterated unboundedly many times. Clearly, there are only finitely many different reachability abstractions (for a fixed number of counters). We can compute them for all pairs of states of an R-automaton by dynamic programming (computing $C_{s, s'}$ with paths restricted to k steps for all pairs of states with increasing k until a fixed point is reached). Then, the non-emptiness problem for infinite words translates to checking whether there is an accepting state $s \in F$ such that $C_{s_0, s} \neq \emptyset$ and there is a $t \in C_{s, s}$ such that $t \in \{0, r\}^n$.

The complexity follows from the fact that the non-deterministic reachability procedure has to remember the value 0, 1, or r for every counter.

7 Conclusions

We have defined R-automata – finite automata extended with unbounded counters which can be left unchanged, incremented, or reset along the transitions. As the main result, we have shown that the following problem is decidable in 2-EXPSpace. Given an R-automaton, is there a bound such that all words are accepted by runs along which the counters do not exceed this bound? We have also extended this result to R-automata with Büchi acceptance conditions.

As a future work, one can consider the (bounded) universality or limitedness question to vector addition systems (VASS) or reset vector addition systems (R-VASS), where the latter form a superclass of R-automata. The limitedness problem can be shown undecidable for R-VASS for both finite word and ω -word case, while it is an open question for VASS. The universality problem can be shown to be undecidable for R-VASS for ω -word case, in other cases it is open.

References

- [AKY07] P. A. Abdulla, P. Krcal, and W. Yi. Sampled universality of timed automata. In *Proc. of FOSSACS'07*, volume 4423 of *LNCS*, pages 2–16. Springer-Verlag, 2007.
- [BC06] Mikolaj Bojańczyk and Thomas Colcombet. Bounds in omega-regularity. In *LICS'06*, pages 285–296. IEEE Computer Society Press, 2006.
- [CL08] Thomas Colcombet and Christof Löding. The non-deterministic Mostowski hierarchy and distance-parity automata. In *ICALP'08*, volume 5126 of *LNCS*, pages 398–409. Springer-Verlag, 2008.
- [Col07] Thomas Colcombet. Factorisation forests for infinite words. In *FCT'07*, volume 4639 of *LNCS*, pages 226–237. Springer-Verlag, 2007.
- [Has82] Kosaburo Hashiguchi. Limitedness theorem on finite automata with distance functions. *Journal of Computer and System Sciences*, 24(2):233–244, 1982.
- [Has90] Kosaburo Hashiguchi. Improved limitedness theorems on finite automata with distance functions. *Theoretical Computer Science*, 72(1):27–38, 1990.
- [Kir05] Daniel Kirsten. Distance desert automata and the star height problem. *Informatique Theorique et Applications*, 39(3):455–509, 2005.
- [KP05] Pavel Krcal and Radek Pelanek. On sampled semantics of timed systems. In *Proc. of FSTTCS'05*, volume 3821 of *LNCS*, pages 310–321. Springer-Verlag, 2005.
- [Leu91] Hing Leung. Limitedness theorem on finite automata with distance functions: an algebraic proof. *Theoretical Computer Science*, 81(1):137–145, 1991.
- [Sim90] Imre Simon. Factorization forests of finite height. *Theoretical Computer Science*, 72(1):65–94, 1990.
- [Sim94] Imre Simon. On semigroups of matrices over the tropical semiring. *Informatique Theorique et Applications*, 28(3-4):277–294, 1994.

Paper III



Sampled Semantics of Timed Automata*

Parosh Aziz Abdulla, Pavel Krcal, and Wang Yi

Department of Information Technology
Uppsala University, Sweden

Email: {parosh,pavelk,yi}@it.uu.se

Abstract. Sampled semantics of timed automata is a finite approximation of their dense time behavior. While the former is closer to the actual software or hardware systems with a fixed granularity of time, the abstract character of the latter makes it appealing for system modeling and verification. We study one aspect of the relation between these two semantics, namely checking whether the system exhibits some qualitative (untimed) behaviors in the dense time which cannot be reproduced by any implementation with a fixed sampling rate. More formally, the *sampling problem* is to decide whether there is a sampling rate such that all qualitative behaviors (the untimed language) accepted by a given timed automaton in dense time semantics can be also accepted in sampled semantics. We show that this problem is decidable.

1 Introduction

Dense time semantics allows timed automata [3] to delay for arbitrary real valued amounts of time. This includes also arbitrarily small delays and delays which differ from each other by arbitrarily small values. Neither of these behaviors can be enforced by an implementation operating on a concrete hardware. Each such an implementation necessarily includes some (hardware) digital clock which determines the least time delay measurable or enforceable by the system.

This observation motivates *sampled semantics* of timed automata, which is a discrete time semantics with the smallest time step fixed to some fraction of 1. In other words, the time delays in a sampled semantics with the smallest step ϵ can be only multiples of ϵ . There are infinitely many different sampled semantics, but any of them allows fewer behaviors of the system than dense time semantics. On the other hand, all of the allowed behaviors in a sampled semantics with the sampling rate (the smallest step) ϵ will be preserved in an implementation on a platform with the clock rate ϵ (and all fractions of ϵ).

*This work has been partially supported by the EU CREDO project.

One of the arguments in favor of using dense time semantics is that one does not have to consider a concrete sampling rate of an implementation in the modeling and analysis phase. Dense time semantics abstracts away from concrete sampling rates by including all of them. Also, it seems adequate to assume that the environment stimuli come at any real time point without our control.

If a concrete timed automaton serves as a system description for later implementation, one might try to find a sampling rate which preserves all qualitative behaviors (untimed words). The restriction to qualitative behaviors is necessary, because any sampling rate excludes infinitely many dense time behaviors. By this we lose the explicit timing information, but many important properties, including implicit timing, are preserved. For instance, if we know that the letter b cannot appear later than 5 time units after an occurrence of the letter a in the dense time model and then there is an untimed word accepted by this automaton where a is followed by b then we know that there is a run where b comes within 5 time units after a .

The problem of our interest can be formalized as follows: decide whether for a given timed automaton there is a sampling rate such that all untimed words accepted by the automaton in dense time semantics are also accepted in sampled semantics with the fixed sampling rate. We call this the *sampling problem* for timed automata.

There are timed automata with qualitative behaviors which are not accepted in any sampled semantics. This relies on the fact that timed automata can force differences between the fractional parts of the clock values to grow. In sampled semantics with the smallest time step fixed to ϵ , the distance can only be increased in multiples of ϵ , which implies that the distance between a pair of clocks can grow at most $1/\epsilon$ times. One more increase would make the fractional parts equal again. A sampling rate ensuring acceptance of an untimed word must induce enough valuations within each clock region in order to accommodate increases of the distances between the fractional parts of clock values along some accepting run. If there is a sequence of untimed words which require smaller and smaller time steps in order to be accepted then any fixed sampling necessarily loses some of these words.

To enforce clock difference growth, a timed automaton has to use strict inequalities $<$ and $>$ in its clock guards. Closed timed automata, i.e., timed automata with only non-strict inequalities \leq and \geq in the guards, can be always sampled with the sampling rate 1. Closed timed automata possess one important property – they are *closed under digitization* [16]. The property "*closed under digitization*" has been defined in [18] and it is connected to our problem in the following sense: if the timed language of a timed automaton is closed under digitization then all (untimed) behaviors of this timed automaton are preserved with $\epsilon = 1$.

The growth of clock value differences corresponds to a special type of memory. When a clock value difference grows three times then there must be at least three different clock value differences smaller than the current one. We show that this memory can be characterized by a new type of counter au-

tomata – with finite state control and a finite number of unbounded counters taking values from the natural numbers. The counters can be updated along the transitions by the following instructions:

- 0: the counter keeps its value unchanged,
- 1: the counter value is incremented,
- r : the counter value is reset to 0,
- copy: the counter value is set to the value of another counter,
- max: under some conditions, the counter value can be set to the maximum of sums of pairs of counters.

The sampling problem can be reformulated for our counter automata as follows. We want to decide whether there is a bound such that all words accepted by the automaton can be accepted also by runs along which all counters are bounded by this bound. This problem was studied earlier as the *limitedness* problem for various types of automata with counters. We show that this problem is decidable for our automata by reducing it to the limitedness problem of a simpler type of automata, R-automata [1].

Related work.

The problem of asking for a sampling rate which satisfies given desirable properties has been studied in [4, 8, 14]. In [4], the authors identify subclasses of timed automata (or, digital circuits which can be translated to timed automata) such that there is always an ε which preserves all qualitative behaviors. The problem of deciding whether there is a sampling rate ensuring language non-emptiness is studied in [8, 14]. Work on digitization of timed languages [18] identifies systems for which verification results obtained in discrete time transfer also to the dense time setting. Digitization takes timing properties into account more explicitly, while we consider only qualitative behaviors. A different approach to discretization has been developed in [9]. This discretization scheme preserves all qualitative behaviors for the price of skewing the time passage. Implementability of systems modeled by timed automata on a digital hardware has been studied in [19, 13, 2]. The papers [19, 13] propose a new semantics of timed automata with which one can implement a given system on a sufficiently fast platform. On the other hand, [2] suggests a methodology in which the hardware platform is modeled by timed automata in order to allow checking whether the system satisfies the required properties on the given platform.

The limitedness problem has been studied for various types of finite automata with counters. First, it has been introduced by Hashiguchi [10] for distance automata (automata with one counter which can be only incremented). Different proofs of the decidability of the limitedness problem for distance automata are reported in [11, 15, 17]. Distance automata were extended in [12] with additional counters which can be reset following a hierarchical discipline resembling parity acceptance conditions. Our automata relax this discipline and allow the counters to be reset arbitrarily. Universality of a similar type of automata for tree languages is studied in [7, 6]. A model with counters which can be incremented and reset in the same way as in R-automata, called B-

automata, is presented in [5]. B-automata accept infinite words such that the counters are bounded along an infinite accepting computation.

Structure of the Paper.

The rest of the paper is organized as follows. In Section 2, we introduce timed automata, dense time and sampled semantics, and our problem. Moreover, we define some technical concepts. Section 3 states the result and sketches the structure of the proof. The model of automata with counters is presented in Section 4, where also the important properties of these automata are shown. The main step of the proof, the construction of a counter automaton from a given timed automaton, together with the correspondence proofs is in Section 5. The proof is completed in Section 6.

2 Preliminaries

In this section, we define syntax and two types of semantics (standard real time and sampled semantics) of timed automata and our problem. We also define region graphs for timed automata and a new notation which simplifies talking about clock differences and clock regions.

Syntax.

Let \mathcal{C} be a finite set of non-negative real-valued variables called *clocks*. The set of guards $G(\mathcal{C})$ is defined by the grammar $g := x \bowtie c \mid g \wedge g$ where $x \in \mathcal{C}, c \in \mathbb{N}_0$ and $\bowtie \in \{<, \leq, \geq, >\}$. A *timed automaton* is a tuple $A = (Q, \Sigma, \mathcal{C}, q_0, E, F)$, where:

- Q is a finite set of locations,
- Σ is a finite alphabet,
- \mathcal{C} is a finite set of clocks,
- $q_0 \in Q$ is an initial location,
- $E \subseteq Q \times \Sigma \times G(\mathcal{C}) \times 2^{\mathcal{C}} \times Q$ is a transition relation, and
- $F \subseteq Q$ is a set of accepting locations.

Semantics.

Semantics is defined with respect to a given time domain \mathbb{T} . We suppose that a time domain is a subset of real numbers which contains 0 and is closed under addition. Also, we suppose that $\mathbb{T} \cap \Sigma = \emptyset$. A *clock valuation* is a function $v : \mathcal{C} \rightarrow \mathbb{T}$. If $r \in D$ then a valuation $v + r$ is such that for each clock $x \in \mathcal{C}$, $(v + r)(x) = v(x) + r$. If $Y \subseteq \mathcal{C}$ then a valuation $v[Y := 0]$ is such that for each clock $x \in \mathcal{C} \setminus Y$, $v[Y := 0](x) = v(x)$ and for each clock $x \in Y$, $v[Y := 0](x) = 0$. The satisfaction relation $v \models g$ for $g \in G(\mathcal{C})$ is defined in the natural way.

The semantics of a timed automaton $A = (Q, \Sigma, \mathcal{C}, q_0, E, F)$ with respect to the time domain \mathbb{T} is a labeled transition system (LTS) $\llbracket A \rrbracket_{\mathbb{T}} = (\hat{Q}, \Sigma \cup \mathbb{T}, \rightarrow, \hat{q}_0)$ where $\hat{Q} = Q \times \mathbb{T}^{\mathcal{C}}$ is the set of states, $\hat{q}_0 = \langle q_0, v_0 \rangle$ is the initial state, $v_0(x) = 0$ for all $x \in \mathcal{C}$. The transition relation is defined as follows: $(\langle q, v \rangle \xrightarrow{\alpha} \langle q', v' \rangle)$ if and only if

- **time step:** $\alpha \in \mathbb{T}$, $q = q'$, and $v' = v + \alpha$, or
- **discrete step:** $\alpha \in \Sigma$, there is $(q, a, g, Y, q') \in E$, $v \models g$, $v' = v[Y := 0]$.

We call paths in the semantics LTS *runs*. Let for a finite run ρ , $l(\rho) \in (\Sigma \cup \mathbb{T})^*$ be the sequence of labels along this path. Let $l(\rho) \upharpoonright \mathbb{T} \in \Sigma^*$ be the sequence of labels with all numbers projected out. We use the same notation also for infinite (countable) runs containing infinitely many discrete steps. Namely, $l(\rho) \upharpoonright \mathbb{T} \in \Sigma^\omega$ if ρ is such a run.

Language.

A finite run $\rho = \langle q_0, v_0 \rangle \longrightarrow^* \langle q, v \rangle$ is accepting if $q \in F$. The (untimed finite word) language of a timed automaton A parameterized with the time domain \mathbb{T} , denoted $L_{\mathbb{T}}(A)$ is the set of words which can be read along the accepting runs of the semantics LTS. Formally, $L_{\mathbb{T}}(A) = \{l(\rho) \upharpoonright \mathbb{T} \mid \rho \text{ is a finite accepting run in } \llbracket A \rrbracket_{\mathbb{T}}\}$.

Let \mathbb{N} denote the set of non-negative integers. An infinite (countable) run with infinitely many discrete steps is accepting if it contains an infinite set of states $\{\langle q, v_i \rangle \mid i \in \mathbb{N}\}$ such that $q \in F$ (standard Büchi acceptance condition). The (untimed) ω -language of a timed automaton A parameterized with the time domain \mathbb{T} , denoted $L_{\mathbb{T}}^\omega(A)$ is the set of words which can be read along the infinite countable accepting runs of the semantics LTS. Formally, $L_{\mathbb{T}}^\omega(A) = \{l(\rho) \upharpoonright \mathbb{T} \mid \rho \text{ is an infinite countable accepting run in } \llbracket A \rrbracket_{\mathbb{T}}\}$.

Let the time domain \mathbb{T}_ε for an $\varepsilon = 1/k$ for some $k \in \mathbb{N}$ be the set $\mathbb{T}_\varepsilon = \{l \cdot \varepsilon \mid l \in \mathbb{N}\}$. We consider the time domains \mathbb{R}_0^+ and \mathbb{T}_ε for all ε . The semantics induced by \mathbb{R}_0^+ is called *dense time semantics* and the semantics induced by a \mathbb{T}_ε is called ε -*sampled semantics*. We use the following shortcut notation: $\llbracket A \rrbracket_{\mathbb{T}_\varepsilon} = \llbracket A \rrbracket_{\mathbb{T}_\varepsilon}$, $L(A) = L_{\mathbb{R}_0^+}(A)$, $L^\omega(A) = L_{\mathbb{R}_0^+}^\omega(A)$, $L_\varepsilon(A) = L_{\mathbb{T}_\varepsilon}(A)$, $L_\varepsilon^\omega(A) = L_{\mathbb{T}_\varepsilon}^\omega(A)$.

Problems.

We deal with the following problems. Decide for a timed automaton A whether there is a $\varepsilon = 1/k$ for some $k \in \mathbb{N}$ such that

- $L_\varepsilon(A) = L(A)$, (sampling)
- $L_\varepsilon^\omega(A) = L^\omega(A)$ (ω -sampling).

Region graph.

We introduce the region equivalence and the standard notion of region graph. Our concept of region equivalence differs from the standard definition in the following technical detail: we consider also the fractional parts of the clocks with the integral part greater than the maximal constant (but we consider only integral parts smaller than or equal to the maximal constant). The important properties of the standard region equivalence (untimed bisimilarity of the equivalent valuations and finite index) are preserved in our definition.

Let for any $r \in \mathbb{R}$, $\text{int}(r)$ denote the integral part of r and $\text{fr}(r)$ denote the fractional part of r . Let k be an integer constant. For a set of clocks \mathcal{C} , the relation \cong_k on the set of clock valuations is defined as follows:

- $v \cong_k v'$ if and only if all the following conditions hold:

- for all $x \in \mathcal{C} : \text{int}(v(x)) = \text{int}(v'(x))$ or $(v(x) > k \wedge v'(x) > k)$,
- for all $x, y \in \mathcal{C} : \text{fr}(v(x)) \leq \text{fr}(v(y))$ if and only if $\text{fr}(v'(x)) \leq \text{fr}(v'(y))$,
- for all $x \in \mathcal{C} : \text{fr}(v(x)) = 0$ if and only if $\text{fr}(v'(x)) = 0$;

Let A be a timed automaton and K be the maximal constant which occurs in some guard in A . For each location $q \in Q$ and two valuations $v \cong_K v'$ it holds that (q, v) is untimed bisimilar to (q, v') . Also, \cong_k has a finite index for all semantics. We call equivalence classes of the region equivalence \cong_K *regions* of A and denote them by D, D', D_1, \dots . For a region D the region D' is the *immediate time successor* if $D' \neq D$, there is $v \in D, r \in \mathbb{R}$ such that $v + r \in D'$, and for all $v \in D, r \in \mathbb{R}$ such that $v + r \in D'$ it holds that $v + r' \in D \cup D'$ for all $r' \leq r$.

Let δ be a letter such that $\delta \notin \Sigma$. Given a timed automaton $A = (Q, \Sigma, \mathcal{C}, q_0, E, F)$, its *region graph* $G = \langle N, \Sigma \cup \{\delta\}, \longrightarrow \rangle$ is a labeled directed graph where the set of nodes N contains pairs $\langle q, D \rangle$, where q is a location of A and D is a region of A and $\longrightarrow \subseteq N \times \Sigma \cup \{\delta\} \times N$ is a set of labeled edges. Informally, the edges lead to an immediate time successor (labeled by δ) or a discrete successor (labeled by a letter from Σ). Formally, $\langle q, D \rangle \xrightarrow{\delta} \langle q, D' \rangle$ if D' is the immediate time successor of D and $\langle q, D \rangle \xrightarrow{a} \langle q', D' \rangle$ if $(l, a, g, Y, l') \in E$, $v \models g$ for all $v \in D$ and $D' = \{v[Y := 0] \mid v \in D\}$.

For a path in the region graph $\sigma = \langle q_1, D_1 \rangle \xrightarrow{w} \langle q_k, D_k \rangle$ we say that a run of the timed automaton in the real or ε -sampled semantics (a path in $\llbracket A \rrbracket_{\mathbb{R}_0^+}$ or $\llbracket A \rrbracket_\varepsilon$, respectively) $\rho = \langle \bar{q}_1, v_1 \rangle \xrightarrow{w} \langle \bar{q}_l, v_l \rangle$ is *along this path* if $k = l$ and for all $1 \leq i \leq k$, $\langle q_i, D_i \rangle$ is the i -th node in σ , $\langle \bar{q}_i, v_i \rangle$ is the i -th state in ρ , $q_i = \bar{q}_i$ and $v_i \in D_i$. We denote this by $\rho \models \sigma$.

By D_ε , where $\varepsilon = 1/k$ for some $k \in \mathbb{N}$, we denote the region D restricted to the valuations from the ε -sampled semantics. I.e., for all $v \in D_\varepsilon$, $v \in D$ and for all clocks x , $v(x) = l \cdot \varepsilon$, where $l \in \mathbb{N}$.

2.1 Notation for Clock Differences and Regions

We introduce the following notation frequently used in Section 5. For two clocks b and d and a clock valuation v , we write \overline{bd}_v to denote the difference between the fractional parts of the clocks b, d in the valuation v . The distance says how much to the right do we have to move the left clock (b in our case), where the movement to the right wraps at 1 back to 0, to reach the right clocks (d in our case). The concept is demonstrated in Figure III.1. This figure depicts a valuation of clocks a, b, c, d , whose integral values we ignore (we can say that they are 0) and whose fractional parts are set according to the figure ($v(a) = 0, v(b) = 0.25, v(c) = 0.55, v(d) = 0.75$). The fractional part of d is greater than that of b and hence to compute \overline{bd}_v we simply record how much do we need to move b to the right to reach d (depicted by the dashed arrow above the solid horizontal line). The fractional part of c is greater than that of b and hence to compute \overline{cb}_v we need to move c to the right until it reaches 1, then it wraps (jumps) to 0, and then we move it further to the right to reach b (depicted by the dashed arrows below the solid horizontal line).

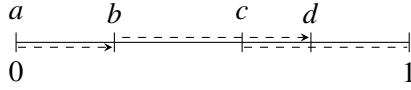


Figure III.1: Illustration of a valuation v and the distances between the fractional parts of the clocks. The values of the clocks are $v(a) = 0, v(b) = 0.25, v(c) = 0.55, v(d) = 0.75$. The distance between b and d is $\overline{bd}_v = 0.3$, the distance between c and b is $\overline{db}_v = 0.5$. Later on, we use this type of diagram only for regions and not for valuations.

Formally, for clocks x, y and a clock valuation v , \overline{xy}_v is defined as follows.

$$\overline{xy}_v = \begin{cases} \text{fr}(v(x)) - \text{fr}(v(y)) & \text{if } \text{fr}(v(x)) \geq \text{fr}(v(y)) \\ 1 - (\text{fr}(v(y)) - \text{fr}(v(x))) & \text{otherwise} \end{cases}$$

We also need to talk about the order of the clocks in a region (an equivalence class of a region equivalence). We say that a region D satisfies an (in)equality $x \bowtie y$ or $x = 0$ (written $x \bowtie_D y, x =_D 0$) where $\bowtie \in \{<, >, =, \leq, \geq, \neq\}$ if it is true for all valuations in the region. Formally, $x \bowtie_D y$ if for all $v \in D$, $\text{fr}(v(x)) \bowtie \text{fr}(v(y))$ and $x =_D 0$ if for all $v \in D$, $\text{fr}(v(x)) = 0$. Note, that for a given region D , either $\text{fr}(v(x)) \bowtie \text{fr}(v(y))$ holds for all the valuations $v \in D$ or it holds for none. Therefore, we adopt the graphical illustration of regions shown in Figure III.2. Here, a region D is depicted, where $\text{fr}(v(a)) = 0, \text{fr}(v(a)) < \text{fr}(v(b)) = \text{fr}(v(e)) < \text{fr}(v(c)) < \text{fr}(v(d))$ for all $v \in D$.

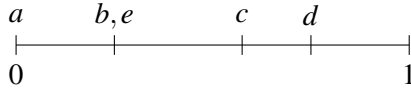


Figure III.2: Illustration of the order of the fractional parts of the clocks in a region D .

The last concept defined here relates the position of three clocks in a region. For clocks x, y, z and a region D , $D \models \overline{xyz}$ tells us that if we start from x and move to the right (and possibly wrap at 1 back to 0), we meet y before we meet z . Formally, $D \models \overline{xyz}$ if there is a time successor D' of D such that $x <_{D'} y$ and $y <_{D'} z$. In Figure III.2, $D \models \overline{bcd}, D \models \overline{cdb}$, holds, but it is not true that, e.g., $D \models \overline{dcb}$.

3 Results

We state the main result of this paper – that our problems are decidable – and sketch the scheme of a proof of this result.

Theorem 1 *Given a timed automaton A , it is decidable whether there is an $\varepsilon = 1/k$ for some $k \in \mathbb{N}$ such that*

- $L_\varepsilon(A) = L(A)$ and

- $L_\varepsilon^\omega(A) = L^\omega(A)$.

First, we claim that this theorem is true for timed automata with less than two clocks. It is trivially true for timed automata without clocks ($|\mathcal{C}| = 0$). In Section 6 we show that for a timed automaton A with only one clock ($|\mathcal{C}| = 1$), $L_{1/2}(A) = L(A)$ and $L_{1/2}^\omega(A) = L^\omega(A)$. We assume that $|\mathcal{C}| \geq 2$ in the rest of the paper.

In Section 4 we develop a tool of independent interest – a non-trivial extension of R-automata. These automata contain unbounded counters which can be incremented, reset to zero, copied into each other, and updated by a special type of max operations. We show that the limitedness problem, i.e., whether there is a bound such that all accepted words can be also accepted by runs along which the counters are smaller than this bound, is decidable for these automata.

The proof of decidability of the sampling problem for timed automata with more than one clock consists of several steps depicted in Figure III.3. We start with a given timed automaton A . The first step is of a technical character. We transform the timed automaton A into an equivalent timed automaton A' with respect to sampling which never resets more than one clock along each transition. In the second step, we build the region graph G for this timed automaton A' . The essential part of the proof is then the third step. Here we transform the region graph G into an extended R-automaton R such that each run in R has a corresponding path in G and vice versa. Moreover, for each run in R and the corresponding path in G , there is a relation between the sampling rate which allows for a concrete run along the path and the maximal counter value along the run. The automaton R operates on an extended alphabet – we have inherited one additional letter δ for time pass transitions from the region graph. In the last step, we remove the transitions labeled by δ and build another extended R-automaton R' such that the timed automaton A' can be sampled if and only if R is limited. This step makes use of the fact that the transitions labeled by δ do not change the counter values, which allows us to use the standard algorithm for removing ε -transitions in finite automata.

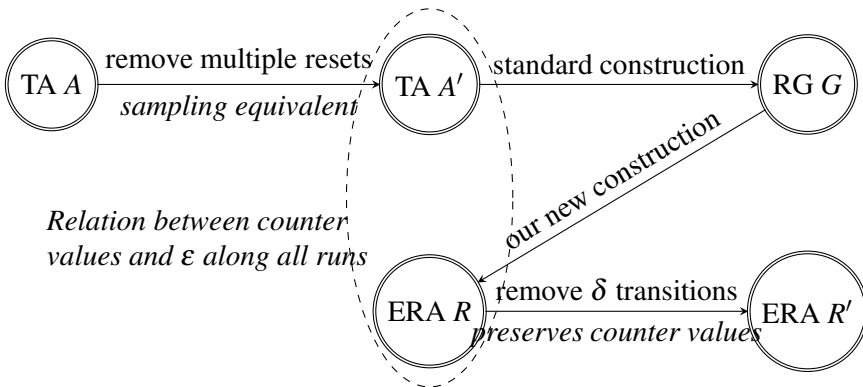


Figure III.3: An overview of the proof structure. The abbreviations TA , RG , and ERA stand for Timed Automaton, Region Graph, and Extended R-Automaton, respectively.

The first and the last step are rather straightforward and we show them in Section 6. The new model of extended R-automata is presented in Section 4. Section 4.3 shows how to reduce the limitedness problem for extended R-automata to the limitedness problem of R-automata, which was shown decidable in [1]. Finally, the main reduction step, the translation of a region graph (induced by A') into an extended R-automaton and the proof of relation between them, together with an informal overview is shown in Section 5.

4 Extended R-automata

In this section we present an extension of R-automata. R-automata are finite state machines with counters which can be updated by the following instructions: no update, increment and reset to zero ($0, 1, r$, respectively). We extend the set of instructions to a copy of one counter value into another counter and taking a maximum of the counters and sums of pairs of counters under specific conditions. For this extension, we show that the limitedness problem is decidable by a reduction to the universality problem of an R-automaton, shown decidable in [1].

4.1 Extensions of R-automata

Before we define syntax and semantics of extended R-automata, we give some informal introduction. The first extension is adding the ability to copy the value of one counter into another counter. The instruction set is extended by instructions $*j$, where j is a counter name and applying this instruction to a counter i results in the counter i having the same value as the counter j .

The other extension we need in order to reduce our problems for timed automata to limitedness of counter automata (taking maxima of counters and counter sums) is rather semantical than syntactical. The only syntactical change is that the reset instruction is equipped with a subset of counters, i.e., if n is the number of counters, reset instructions are $r(A), A \subseteq \{1, \dots, n\}$. The semantics maintains three values for each counter (P, M, N) and a preorder \lesssim on the counters. This rather nonstandard terminology – a counter containing three values – makes the definitions in this section and proofs in Section 4.3 simpler. One can see this as if for a counter i we now have three new counters P_i, M_i , and N_i .

The values N_i behave in the same way as for R-automata with copying. The preorder tells us how to apply the max operation to the values P and M . These values of a counter j are always greater than these values of a counter i such that $i \lesssim j$. More concretely, if $i \lesssim j$ then $M_j \geq M_i + 1$ and if $k, l \lesssim j$ then $P_j \geq P_k + P_l$. The way in which we update the preorder \lesssim along the transitions ensures that, informally, for all counters i , the values P_i and M_i cannot grow unbounded along a run where N_i is bounded.

Syntax.

Let for a given number n of counters, $\mathcal{E} = \{0, 1\} \cup \{r(A) \mid A \subseteq \{1, \dots, n\}\} \cup \{*m \mid 1 \leq m \leq n\}$ be the set of instructions on a counter.

An *extended R-automaton* with n counters is a 5-tuple $R = (S, \Sigma, \Delta, s_0, F)$ where

- S is a finite set of states,
- Σ is a finite alphabet,
- $\Delta \subseteq S \times \Sigma \times \mathcal{E}^n \times S$ is a transition relation,
- $s_0 \in S$ is an initial state, and
- $F \subseteq S$ is a set of final states.

Transitions are labeled (together with a letter) by an effect on the counters. The symbol 0 corresponds to leaving the counter value unchanged, the symbol 1 represents an increment, the symbol $r(A)$ represents a reset (the function of A will be explained later), and a symbol $*j$ means that the value of this counter is set to the value of the counter j . The instructions 0, 1, and $r(A)$ take place first and after that the values are copied. An automaton which does not contain any copy instruction and all resets contain an empty set is called an R-automaton (effects contain only 0, 1, $r(\emptyset)$). We skip the subset of counters A and write r instead of $r(A)$ when the set does not play any role (e.g., in the whole Section 4.2).

We use t, t', t_1, \dots to denote elements of \mathcal{E}^n which we call *effects*. By $\pi_i(t)$ we denote the i -th projection of t . Without loss of generality, we assume that the value of a counter is never directly copied into itself ($\pi_i(t) \neq *i$). A *path* is a sequence of transitions $(s_1, a_1, t_1, s_2), (s_2, a_2, t_2, s_3), \dots, (s_m, a_m, t_m, s_{m+1})$, such that $\forall 1 \leq i \leq m. (s_i, a_i, t_i, s_{i+1}) \in \Delta$. We use s_i to refer to the i -th state of the path. An example of an extended R-automaton is given in Figure III.4.

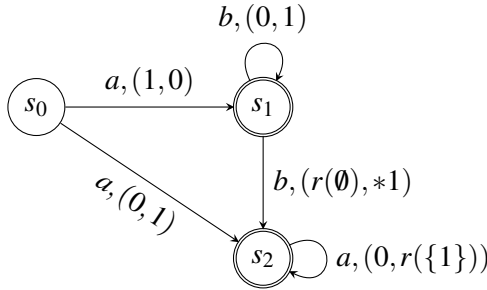


Figure III.4: An R-automaton with two counters.

Unparameterized semantics.

We define an operation \oplus on the counter values: for any $k \in \mathbb{N}$, $k \oplus 0 = k$, $k \oplus 1 = k + 1$, and $k \oplus r = 0$. We extend this operation to n -tuples and copy instructions as follows. For a $t \in \mathcal{E}^n$, let \hat{t} be an effect with all copy instruction replaced by 0, i.e., $\pi_i(\hat{t}) = \pi_i(t)$ if $\pi_i(t) \in \{0, 1, r(A)\}$ and $\pi_i(\hat{t}) = 0$ otherwise. For a $t \in \mathcal{E}^n$ and $(c_1, \dots, c_n) \in \mathbb{N}^n$, $(c_1, \dots, c_n) \oplus t = (c'_1, \dots, c'_n)$, where $c'_i =$

$c_j \oplus \pi_j(\hat{t})$ if $\pi_i(t) = *j$ for some j and $c'_i = c_i \oplus \pi_i(t)$ otherwise. For example, $(1, 5, 7) \oplus (1, *1, *2) = (2, 2, 5)$ – first we increment the first counter and then we copy the values of the first and the second counter into the second and the third counter, respectively.

The operational semantics of an extended R-automaton $R = (S, \Sigma, \Delta, s_0, F)$ is given by an LTS $\llbracket R \rrbracket = (\hat{S}, \Sigma, T, \hat{s}_0)$, where the set of states \hat{S} contains triples $\langle s, \bar{C}, \lesssim \rangle$, $s \in S, \bar{C} \in \mathbb{N}^n \times \mathbb{N}^n \times \mathbb{N}^n$, \lesssim is a preorder on $\{1, \dots, n\}$, with the initial state $\hat{s}_0 = \langle s_0, \bar{C}_0, \emptyset \rangle$, where $\bar{C}_0 = (0^n, 0^n, 0^n)$. For a $\bar{C} \in \mathbb{N}^n \times \mathbb{N}^n \times \mathbb{N}^n$, we denote the first projection by \bar{P} , the second projection by \bar{M} , and the third projection by \bar{N} . I.e., $\bar{P}, \bar{M}, \bar{N} \in \mathbb{N}^n$ and $\bar{C} = (\bar{P}, \bar{M}, \bar{N})$. For $1 \leq i \leq n$, we denote by P_i, M_i , or N_i the i -th projection of \bar{P}, \bar{M} , or \bar{N} , respectively. The role of the preorder \lesssim and of the counter valuation is informally explained below the formal definition of the transition relation. We introduce a shorthand $i \simeq j$ for $(i \lesssim j \wedge j \lesssim i)$ and $i \not\lesssim j$ for $(i \lesssim j \wedge \neg(j \lesssim i))$.

The transition relation is defined as follows: $(\langle s, \bar{C}, \lesssim \rangle, a, \langle s', \bar{C}', \lesssim' \rangle) \in T$ if and only if $(s, a, t, s') \in \Delta$ and \bar{C}', \lesssim' are constructed by the following three steps (executed in this order):

1. $\bar{P}' = \bar{P} \oplus t, \bar{M}' = \bar{M} \oplus t$, and $\bar{N}' = \bar{N} \oplus t$
2. The preorder \lesssim' is constructed in two steps. First, $i \lesssim' j$ if and only if either:
 1. $i \lesssim j$ and $\pi_i(t) \in \{0, 1\}, \pi_j(t) \in \{0, 1\}$ and it is not true that $j \lesssim i, \pi_i(t) = 1, \pi_j(t) = 0$, or
 2. $\pi_i(t) = r(\{j\} \cup A)$ and $N'_j > 0$, or
 3. $\pi_i(t) = *j$ or $\pi_j(t) = *i$.
Secondly, add the transitive and reflexive closure to \lesssim' .
4. Repeat the following until a fixed point is reached: if $i \not\lesssim j$ then set $M'_j = \max\{M'_j, M'_i + 1\}$ and if $k, l \not\lesssim j$ then $P'_j = \max\{P'_j, P'_k + P'_l\}$.

We shall call the states of the LTS *configurations*. We write $\langle s, \bar{C}, \lesssim \rangle \xrightarrow{a} \langle s', \bar{C}', \lesssim' \rangle$ if $(\langle s, \bar{C}, \lesssim \rangle, a, \langle s', \bar{C}', \lesssim' \rangle) \in T$. We extend this notation also for words, $\langle s, \bar{C}, \lesssim \rangle \xrightarrow{w} \langle s', \bar{C}', \lesssim' \rangle$, where $w \in \Sigma^+$.

Note that the values \bar{N} of the counters are updated only by the instructions $0, 1, r(A)$ and $*j$ (Step 1). The values \bar{M} and \bar{P} of the counters are updated by these effects as well (Step 1), but they can also be increased by the max function (Step 4). Namely, $N_i > 0$ implies that $M_i > 0$ and $P_i > 0$. Clearly, there is always a fixed point reached after at most n iterations of Step 4.

The preorder \lesssim in a reachable state $\langle s, \bar{C}, \lesssim \rangle$ relates counters i, j only if the values M_i, P_i are smaller than or equal to M_j, P_j ($i \lesssim j$ implies $M_i \leq M_j, P_i \leq P_j$). Especially, $i \simeq j$ implies $M_i = M_j, P_i = P_j$. This is satisfied in the initial state (trivially) and preserved by updates in Step 2. There, the effects influence the preorder \lesssim in the following way: an equality is broken if one counter is incremented and the other one is left unchanged (Step 1), a reset removes the counter from the preorder and puts it below non-zero counters indicated in the reset (Step 2), and a copy instruction sets the counter equal to the counter whose value it copied (Step 3). In other cases, the relation is

preserved (Step 1). An example of the effect of Step 2 on a preorder is in Figure III.5.

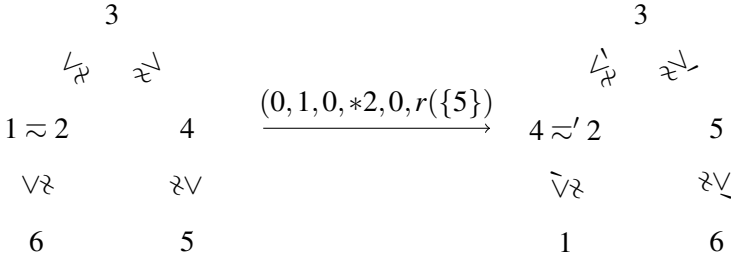


Figure III.5: An example of updates of \lesssim after applying the effect $(0, 1, 0, *2, 0, r(\{5\}))$. The diagram on the left side depicts \lesssim and the diagram on the right side depicts \lesssim' . Step 1 sets $1 \lesssim' 3, 2 \lesssim' 3, 5 \lesssim' 3, 1 \lesssim' 2$. It does not set $2 \lesssim' 1$, because the counter 2 was incremented while the counter 1 was left unchanged. Step 2 sets $6 \lesssim' 5$. Step 3 sets $4 \lesssim' 2, 2 \lesssim' 4$. The transitive and reflexive closure completes \lesssim' to a preorder.

Another view on the preorder is what sequence of effects results in $i \lesssim j$. This can happen only in the following three ways. First, when i is reset with j in the set, i.e., by $r(\{j\} \cup A)$, and $M_j > 0$. Second, i is copied to j or j is copied to i and then j is incremented by 1 while i stays unchanged (the instruction is 0). Third, the relation $i \lesssim j$ can also be a result of the transitive closure. If already $i \lesssim j$ holds then it can be broken only by a reset or a copy of one of these two counters.

The preorder \lesssim influences only the values \bar{P} and \bar{M} . If we skip Step 2 in the semantics (which would result in \lesssim to be empty in all the reachable states) then $\bar{P} = \bar{M} = \bar{N}$ in all the reachable states. Also, changes of the values \bar{N} along a transition depend only on the effect and not on \lesssim in the starting state.

We could also view our extension as R-automata which can perform max operations on the counters along the transitions. The motivation for introducing the preorder \lesssim instead of allowing explicit max operations as instructions on the transitions is to restrict the usage of max operations so that Lemma 7 and Lemma 8 hold. Unrestricted usage of max operation is equivalent to alternation. Limitedness has been shown decidable for alternating cost tree automata in [6], but resets have to follow a hierarchical (parity-like) discipline in these automata and copying is not allowed.

Paths in an LTS are called *runs* to distinguish them from paths in the underlying extended R-automaton. Observe that the LTS contains infinitely many states, but the counter values do not influence the computations, since they are not tested anywhere. In fact, for any extended R-automaton R , $\llbracket R \rrbracket$ is bisimilar to R considered as a finite automaton (without counters and effects).

Parameterized Semantics.

Next, we define B -semantics of extended R-automata. The parameter B is a bound on the counter values \bar{N} which can occur along any run. For a given $B \in$

\mathbb{N} , let \hat{S}_B be the set of configurations restricted to the configurations which do not contain a counter whose \bar{N} values exceed B , i.e., $\hat{S}_B = \{\langle s, \bar{C}, \bar{\lesssim} \rangle \mid \langle s, \bar{C}, \bar{\lesssim} \rangle \in \hat{S} \wedge \bar{C} = (\bar{P}, \bar{M}, \bar{N}) \wedge \forall 1 \leq i \leq n. N_i \leq B\}$. For an extended R-automaton R , the B -semantics of R , denoted by $\llbracket R \rrbracket_D$, is $\llbracket R \rrbracket$ restricted to \hat{S}_B . We write $\langle s, \bar{C}, \bar{\lesssim} \rangle \xrightarrow{a}_B \langle s', \bar{C}', \bar{\lesssim}' \rangle$ to denote the transition relation of $\llbracket R \rrbracket_D$. We extend this notation also for words, $\langle s, \bar{C}, \bar{\lesssim} \rangle \xrightarrow{w}_B \langle s', \bar{C}', \bar{\lesssim}' \rangle$, where $w \in \Sigma^+$.

Language.

The (unparametrized or B -) language of an extended R-automaton is the set of words which can be read along the runs in the corresponding LTS ending in an accepting state (a configuration whose first component is an accepting state). Formally, for a run ρ in $\llbracket R \rrbracket$, let $l(\rho)$ denote the concatenation of the labels along this run. A run $\rho = \langle s_0, \bar{C}_0, \emptyset \rangle \xrightarrow{*} \langle s, \bar{C}, \bar{\lesssim} \rangle$ is accepting if $s \in F$. The *unparametrized language* accepted by an extended R-automaton R is $L(R) = \{l(\rho) \mid \rho \text{ is an accepting run in } \llbracket R \rrbracket\}$. For a given $B \in \mathbb{N}$, the B -language accepted by an extended R-automaton R is $L_B(R) = \{l(\rho) \mid \rho \text{ is an accepting run in } \llbracket R \rrbracket_D\}$. The unparametrized language of the extended R-automaton from Figure III.4 is ab^*a^* . The 2-language of this automaton is $a(\varepsilon + b + bb + bbb)a^*$. We also in the standard way define the language of infinite words for R-automata with Büchi acceptance conditions, denoted by $L^\omega(R), L_B^\omega(R)$.

Limitedness/Universality.

The language of an extended R-automaton R is *limited* or *universal* if there is a natural number B such that $L_B(R) = L(R)$ or $L_B(R) = \Sigma^*$, respectively. The definition of these problems for ω -languages is analogous. We show in Lemma 2 that it is decidable whether a given extended R-automaton is limited or universal and in Lemma 7 and Lemma 8 that this concept would not change even if we limit the \bar{P} or \bar{M} values in the definition of B -semantics.

We could split an extended R-automaton into three different automata which would maintain only one of the values $\bar{P}, \bar{M}, \bar{N}$. Later on, in the reduction from timed automata to these automata, we use only \bar{P} values. The presentation which we chose (all values together in one automaton) simplifies the notation for the proofs of Lemma 7 and Lemma 8.

4.2 Limitedness of Extended R-automata – Copy Operations

First, we show that the limitedness problem for extended R-automata is decidable. In this section, we deal only with the N values of extended R-automata. We ignore the preorder $\bar{\lesssim}$ (as it is not needed for calculating the N values) and when we say that a counter i has a value k then we mean that $N_i = k$. We also write only r instead of $r(A)$. The decidability proof reduces the limitedness problem for extended R-automata to the limitedness problem of R-automata. It has been show in [1] that the universality problem of R-automata is decidable, but it is easy to see that this procedure can be used also to decide the

limitedness problem. We create a disjunct union of the R-automaton in question and its complement (where the automaton is considered without effects, as a standard finite automaton). We add effects $(0, \dots, 0)$ on all transitions of the complement. This automaton is universal if and only if the original R-automaton is limited.

Lemma 2 *For a given extended R-automaton R , the questions whether there is $B \in \mathbb{N}$ such that $L_B(R) = L(R)$ (and $L_B^\omega(R) = L^\omega(R)$) is decidable.*

The rest of this subsection proves this lemma. In order to avoid unnecessary technical complications in the main part of the proof, we restrict ourselves to extended R-automata with at most one copy instruction in each effect. We show how to extend the proof to the general model at the end of this section. We reduce the universality problem for extended R-automata to the universality problem of R-automata, for which it has been shown that this problem is decidable [1].

Construction.

As the first step, we equip each R-automaton with a variable called *parent pointer* for each counter and with the ability to swap the values of the counters. The parent pointers range over $\{\text{null}\} \cup \{1, \dots, n\}$, where n is the number of the counters. Later on we use them to capture (a part of) the history of copying. We observe that for each R-automaton one can encode the value swapping and the parent pointers into the states. To express the encoding more formally, let us assume that the transitions in the semantics LTS are labeled also by the counter values (in the order encoded by the automaton) and the parent pointers. For each R-automaton \hat{R} with parent pointers and value swapping, we can build an R-automaton \bar{R} with $|S| \cdot n! \cdot 2^n$ states bisimilar to \hat{R} , where $|S|$ is the number of the states of \hat{R} . Moreover, any number of value swaps and parent pointer operations can be encoded along each transition of \bar{R} together with standard updates (increments, resets). \bar{R} can also branch upon the values of the parent pointers.

Now we can present the reduction by constructing an R-automaton \hat{R} which uses counter value swapping and the parent pointers for each extended R-automaton R such that \hat{R} is limited if and only if R is limited. \hat{R} has all the states of R together with an error sink and it has the same initial state s_0 and the same set of accepting states as R . The error sink is a non-accepting state with no outgoing transitions except for the self-loops labeled by Σ and the effect $(0, \dots, 0)$ which do not swap any counter values and do not manipulate the parent pointers. The automaton starts in the initial state with all parent pointers set to null. To define the transitions of \hat{R} , we need to encode the copying using the tools of R-automata with parent pointers and value swapping. To do this, we replace each copy by a reset, possibly with some (non-deterministic) value swapping and bookkeeping of the parent pointers. The parent pointers will help us to check whether all the non-deterministic choices were done correctly.

For each transition of R we either construct simulating transitions or a transition going to the error sink. Let us denote the simulated transition of R by

$s \xrightarrow{a,t} s'$, where $t = (e_1, \dots, e_n)$. If there are counters k, l such that $e_k \in \{0, 1\}$, $e_l \notin \{0, 1\}$, and the parent pointer of k points to (is set to) l then we create a transition going to the error sink. Otherwise, we build simulating transitions $s \xrightarrow{a,t',sp} s'$ in \hat{R} labeled by an effect $t' = (e'_1, \dots, e'_n)$, which might also swap some counter values and manipulate the parent pointers (denoted by sp).

If t does not contain any copy instruction then there is one simulating transition with $t' = t$ and for all i such that $e_i = r$, we set i 's parent pointer to null. No counter values are swapped.

If t contains a copy instruction $e_i = *j$ then we create two simulating transitions. Each of them has the same effect $t' = (e'_1, \dots, e'_n)$, where $e'_k = e_k$ if $k \neq i$ and $e_i = r$. These two transitions give the simulating automaton a non-deterministic choice between the counters i and j . The first transition corresponds to the choice of j . Along this transition, we perform the effect and set i 's parent pointer to j . No counter values are swapped. Along the other transition (corresponding to the choice of i), we perform the effect, swap the values of the counters i and j , we copy the value of j 's parent pointer into i 's parent pointer, we change the value of all parent pointers with value j to i , and finally we set j 's parent pointer to i . Both transitions also set the k 's parent pointer to null for all k such that $e_k = r$. An example of the construction of simulating transitions for a transition with an effect containing a copy instruction is depicted in Figure III.6.

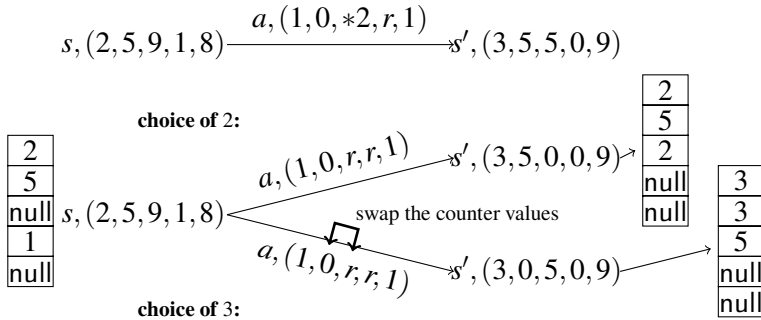


Figure III.6: An example of the construction of simulating transitions for a transition from s to s' labeled with an effect $(1, 0, *2, r, 1)$. In this example, the simulating transitions start from the state s with the parent pointers set to the values 2, 5, null, 1, null (for the counters 1, 2, ..., 5) and the counter values set to (2, 5, 9, 1, 8). The parent pointer values and the counter values only illustrate the parent pointer manipulations and the application of the effects, they might differ in actual runs.

Proof of Correctness.

Intuitively, the choice of a counter in the copy instruction tells that the value in this counter will be destroyed by a reset or overwritten by a copy instruction later than in the counter which was not chosen. The structure of the copies is captured by the parent pointers in the following sense. If the counter i points to the counter j then i contains an immediate copy of j (but possibly modified

by increments) and its value will be destroyed earlier than the value in j . The automaton ends in the error sink if it witnesses a violation of some of these implicit claims, i.e., the value in the counter i is destroyed earlier than the value in the counter j .

First, we formalize the concept of the evolution of a value and define the corresponding runs. Then we show existence of corresponding accepting runs. Later on we use the fact that the parent pointers along the simulating traces have a special structure to show the correctness of the simulation.

Definition 1 *For a path σ of length $|\sigma|$ in the extended R-automaton (considered as a graph) with n counters and for two natural numbers $1 \leq i < j \leq |\sigma|$, a total function $vt : \{i, i+1, \dots, j\} \rightarrow \{1, \dots, n\}$ is a value trace if for all k such that $i \leq k < j$, t is the effect on the transition between the k -th and $k+1$ -st state on σ , $vt(k) = a$, $vt(k+1) = b$, the following holds: if $a \neq b$ then $\pi_b(t) = *a$ and if $a = b$ then $\pi_b(t) \in \{0, 1\}$.*

A value trace follows a value from some time point during its evolution (increments, copying) in an extended R-automaton. A value trace ends before the value is overwritten by a copy instruction or reset. We also talk about a value trace along a run. Then we mean a value trace along a path which has induced the run. We order value traces by the set inclusion on their domains (e.g., $vt : \{2, 3\} \rightarrow \{1, \dots, n\}$ is smaller than $vt : \{2, 3, 4\} \rightarrow \{1, \dots, n\}$ regardless of the actual function values). We define the *length* of a value trace as the size of its domain.

Now we define the correspondence between accepting runs in an extended R-automaton R and in its corresponding R-automaton \hat{R} . We say that a run ρ of R over w and a run ρ' of \hat{R} over w are corresponding if for all i the i -th transitions of ρ, ρ' are obtained by executing the transitions $s \xrightarrow{a,t} s'$ and $s \xrightarrow{a,t',sp} s'$, where $s \xrightarrow{a,t} s'$ is a simulating transition of $s \xrightarrow{a,t} s'$. We show that for each accepting run of one automaton there is an accepting corresponding run of the other automaton. It follows immediately from the definitions that for each accepting run of \hat{R} there is exactly one accepting corresponding run of R .

The other direction is more complicated, because we have to show that \hat{R} can choose correct values for non-deterministic choices in the copy instruction so that it does not end up in the error sink. For each accepting run ρ of R , we construct an accepting run ρ' of \hat{R} as follows. We label each counter j in the k -th state of ρ (for all $k \leq |\rho|$) by the length of a maximal value trace vt with domain being a subset of $\{k, k+1, \dots, |\rho|\}$ and $vt(k) = j$ (this label is called *expectancy*). \hat{R} takes the simulating transition for each transition of ρ (according to the rules above) and when it has to choose between i and j ($e_i = *j$) along a transition ending in the k -th state, then it chooses i if and only if the expectancy of i in k is greater than the expectancy of j in k (*expectancy rule*). We show that this is a valid definition, i.e., the corresponding run of \hat{R} does not end up in the error sink. The main step in the proof is to show that the parent pointers always point to the counters with expectancy which is greater than or equal to the expectancy of the counter which owns the parent pointer.

Lemma 3 *For each accepting run ρ of R there is an accepting corresponding run ρ' of \hat{R} .*

Proof. We prove by induction that for each prefix of ρ there is a simulating run which does not contain the error state such that for any state along ρ' and any two counters i, j in this state, if the parent pointer of i points to j then the expectancy of j is not smaller than that of i . Such a simulating run for $|\rho|$ will also be accepting.

The basic step (i.e., the prefix length is 0) is trivial. For the induction step, let us assume that there is a simulation of the prefix of length k satisfying IH. To simulate the $k + 1$ -st transition, we follow the expectancy rule.

Because of the induction hypothesis and the definition of expectancy, there are always simulating transitions (and not a transition leading to the error sink). If there is a copy instruction $e_i = *j$ in the transition, the non-deterministic choice is performed according to the vt function, so the result again satisfies the induction hypothesis. The transfer of the parent pointers does not violate it either, because expectancy of j in k is equal to 1 plus the maximum of the expectancies of i and j in $k + 1$. The resets do not establish any new parent pointers, so the result again satisfies the induction hypothesis. The other instructions result in decrementing the expectancy, which preserves the induction hypothesis for all the pointers inherited from the previous state as well as for the pointers changed by the copy instruction. ■

Let us introduce the parent pointer relation \rightarrow_p for a state of \hat{R} as a relation on counters where $i \rightarrow_p j$ if and only if the parent pointer of i is set to j .

Lemma 4 *Let ρ be a run of \hat{R} . The transitive closure of \rightarrow_p is antireflexive in all states of ρ .*

Proof. We prove by induction that for each prefix of ρ , the transitive closure of \rightarrow_p is antireflexive in all states of the prefix.

The basic step is trivial, \rightarrow_p is empty in s_0 . For the induction step, we need to check that a single transition does not violate the antireflexivity. If the transition leads to the error sink then \rightarrow_p is not changed. Otherwise, it is a simulating transition defined by the rules above. The resets make \rightarrow_p smaller and 0, 1 do not change it. In the copy instruction $e_i = *j$, we introduce one new pointer, but we know that nothing points to i , because of the condition on creating the simulating transitions and the fact that the parent pointers of all reset counters are set to null. In the first case (j has been chosen), we set i 's parent pointer to j , which cannot introduce a loop, since nothing points to i . In the second case (i has been chosen), since we have redirected all the pointers pointing to j to i , there is nothing pointing to j and newly introduced $j \rightarrow_p i$ cannot create a loop. Also, since there was nothing pointing to i previously, the only pointers pointing to i now are those that previously pointed to j . ■

This leads to the following definition of ranks. For a counter i in a state s of \hat{R} we define $\text{rank}(s, i)$ inductively by $\text{rank}(s, i) = 0$ if the parent pointer of i in s is null and $\text{rank}(s, i) = \text{rank}(s, j) + 1$ if $i \rightarrow_p j$ in s . From Lemma 4, we

have that the ranks are well-defined and it follows directly from the definition that the rank of a counter is always bounded by the number of the counters. Now we formulate a lemma saying that the ranks never decrease along a value trace.

Lemma 5 *Let ρ be a run of \hat{R} and vt be a value trace. Then for $k \leq l$ such that $vt(k), vt(l)$ are defined, $\text{rank}(s_k, vt(k)) \leq \text{rank}(s_l, vt(l))$.*

Proof. We show this claim by induction on $l - k$. The basic step is that $l = k$ and then $\text{rank}(s_k, vt(k)) = \text{rank}(s_l, vt(l))$. For the induction step we have two cases. If the transition leads to the error sink then \rightarrow_p is not changed and therefore the ranks do not decrease. Otherwise, it is a simulating transition defined by the rules above. Because of the condition on creating the simulating transitions, we never decrease any rank by a reset. The instructions 0, 1 also do not decrease any rank. Copy increases the rank of the branch with smaller expectancy (and the counter is reset) and keeps the rank for the branch with bigger expectancy (the one which keeps the value) unchanged. Because of the careful manipulation with the pointers, no ranks which depend on the rank of the longer branch change either. ■

The main property of the reduction is stated in the following lemma. The correctness of Lemma 2 is then a direct corollary of this lemma.

Lemma 6 *Let R be an extended R-automaton with n counters and with at most one copy instruction in each effect and \hat{R} be the simulating R-automaton constructed as above. For each B and for each word w , $w \in L_B(R) \Rightarrow w \in L_B(\hat{R})$ and $w \in L_B(\hat{R}) \Rightarrow w \in L_{n \cdot B}(R)$.*

Proof. The first implication: we know from Lemma 3 that for each accepting run ρ of R over w there is a corresponding accepting run ρ' of \hat{R} over w . It follows directly from the construction that for all $k \leq |\rho|$, the counter values in the k -th state of ρ' are bounded by the counter values in the k -th state of ρ . All instructions are simulated faithfully except for replacing copy instructions by resets along ρ' .

The second implication: by contraposition, let us for each B consider a word w such that $w \notin L_{n \cdot B}(R)$. Any accepting run ρ' of \hat{R} over w must satisfy Lemma 5. Let vt be a maximal value trace for a value which exceeds $n \cdot B$ in ρ' . We study the evolution of this value in ρ' . It is simulated faithfully except for some possible resets in the copy instructions. But for each such reset, the rank of the counter strictly increases. Therefore, there can be at most $n - 1$ such resets and there must be a state in which this value exceeds B . ■

Now we show that the result holds also for extended R-automata with any number of copying in each step. Let us view the relation " i is copied to j " induced by an effect t as a directed graph (counters are nodes, there is an edge from i to j if $\pi_j(t) = *i$). Because each node can have at most one incoming edge, such a graph is a collection of simple loops with isolated paths outgoing from them (nodes with no incoming edge are considered as degenerated loops). We can split application of such an effect t into an equivalent sequence of effects with at most one copy instruction and some swapping of the values

and the parent pointers as follows. First, we perform \hat{t} (all increments and resets). Then we pick one of the counters j such that j has no outgoing edge and it has an (exactly one) incoming edge from i . We copy the value of i to j and leave all other counters unchanged, which can be described by the effect $(0, \dots, *i, \dots, 0)$, where $*i$ is on the j -th position. Then we remove the edge connecting i and j and continue to pick another such counter. When there is no node j with no outgoing edge and with an incoming edge, there still might be loops in the copying graph. We simply swap the counter values and the parent pointers in the loops. Because of the order in which we have copied the counters, the effect of this sequence of transitions with at most one copy instruction and swaps is the same as that of the original transition. Also, the correctness does not depend on the order in which we choose the edges. A careful analysis shows that this sequence of transitions can be encoded into one simulating transition in R-automata with value swapping and parent pointers.

4.3 Limiting Maxima in Extended R-automata

Let for a state $\langle s, (\bar{P}, \bar{M}, \bar{N}), \lesssim \rangle$ in a run of an extended R-automaton with n counters, the N -value (M -value, P -value) of this state be $\max\{N_i | 1 \leq i \leq n\}$ ($\max\{M_i | 1 \leq i \leq n\}$, $\max\{P_i | 1 \leq i \leq n\}$, respectively). Let for a run ρ of this automaton, the N -value (M -value, P -value) of the run be the maximum state N -value (M -value, P -value) over all states along the run. We denote this value by $N(\rho)$ ($M(\rho)$, $P(\rho)$).

Lemma 7 *Let R be an extended R-automaton with n counters and let $B \in \mathbb{N}$. For all runs ρ of R , if $N(\rho) \leq B$ then $M(\rho) \leq B^n$.*

Proof. We show a stronger claim, namely that if a run ρ starts in a state with the M -value equal to b and $N(\rho) \leq B$ then $M(\rho) \leq b + B^n$, by induction on the number of counters n . The basic step ($n = 1$) is trivial, because Step 4 will never change the counter value and thus $M(\rho) = N(\rho) \leq B$.

Let us assume that the claim holds for automata with n counters. We show that it holds for automata with $n + 1$ counters. Let us fix a run ρ and a $B \in \mathbb{N}$. Let us without loss of generality assume that the counter which reaches the greatest M value is the counter $n + 1$. First, we argue that there is an extended R-automaton and a run of this automaton starting with the same counter values as ρ which has the same M -value as ρ , along which the counter $n + 1$ is never updated by a copy instruction and never reset.

The argument for the copy instructions is straightforward, each copy instruction makes the source and the target counter equivalent both in the values which it contains (Step 1) and in the preorder \lesssim (Step 3). Therefore, we can permute the instructions in the effects (intuitively, rename the counters) in the prefix of the run leading to the copy instruction so that the value is accumulated in the counter $n + 1$ and then copied to the other counter.

If the counter $n + 1$ is reset then its values can be incremented only by 1 and via the max operation with other counters which are reset later. This follows from the fact that $n + 1$ is a minimal element of \lesssim after it is reset. This is the

same situation as if the run started with all counter values equal to zero (\bar{C}_0) and \lesssim empty.

Therefore, the counter $n + 1$ can be updated only by 1 and 0 (where 0 does not increase M_{n+1} and there can be at most B increases by 1) and M_{n+1} can be increased by the max operation. We show that M_{n+1} can grow by at most B^n between any two increments by 1.

Between any two increments by 1, the value M_{n+1} can grow only by application of the max function with the counters i such that $i \lesssim n + 1$ (Step 4). These counters cannot make use of the counter $n + 1$ (cannot increase their M -values more than if there was no counter $n + 1$). The only way for a counter i to use the counter $n + 1$ is to apply $i = *(n + 1)$, but this would set $i \simeq n + 1$ (Step 3). To set $i \lesssim n + 1$ back again, we would have to reset i (instruction $r(\{n + 1\} \cup A)$) or copy some other counter j such that $j \lesssim n + 1$ into i (instruction $i = *j$) (follows from Step 2). But this would have the same effect as if i was updated by 0 until this state and then reset or copied. Hence, the claim that M_{n+1} can grow by at most B^n between any two increments by 1 follows from IH. ■

Now we show that the P -values are bounded by an exponent of the M -values.

Lemma 8 *Let R be an extended R -automaton with n counters and let $B \in \mathbb{N}$. For all runs ρ of R , if $M(\rho) \leq B$ then $P(\rho) \leq 2^B$.*

Proof. We show by induction on the length of the run that for all states $\langle s, (P, M, N), \lesssim \rangle$ along the run and for all $1 \leq i \leq n$, $P_i \leq 2^{M_i}$. The basic step is trivial. We check that the claim is preserved by every update of the counters. Let us denote the values before the transition by unprimed letters P, M and after the effect takes place with primed letters P', M' . Let the instruction (update) applied to the counter i be:

- 0 : The values of the counters do not change, the claim holds from IH.
- 1 : We have that $P'_i = P_i + 1$, $M'_i = M_i + 1$. From IH, we know that $P_i \leq 2^{M_i}$. From this we have that $P'_i = P_i + 1 \leq 2^{M_i} + 1$. Because $2^{M_i} \geq 1$ for all $M_i \geq 0$, we have that $2^{M_i} + 1 \leq 2 \cdot 2^{M_i} = 2^{M_i+1} = 2^{M'_i}$.
- r : This case is clear, $P'_i = M'_i = 0$.
- *j : The claim follows from IH.
- max : Let us discuss one application of the max function (Step 4) where the value of P'_i is increased (if it is not the case then the claim holds from IH). If $k, l \lesssim i$ then $P'_i = \max\{P_i, P_k + P_l\} = P_k + P_l$ and $M'_i = \max\{M_i, M_k + 1, M_l + 1\}$. Without loss of generality, let us assume that $P_k \geq P_l$. Thus, $P'_i \leq 2 \cdot P_k \leq 2 \cdot 2^{M_k} = 2^{M_k+1}$. Since $M'_i \geq M_k + 1$, we have that $2^{M_k+1} \leq 2^{M'_i}$. Update of all counters along each transition consists only of these updates. ■

5 Encoding of Timed Automata to Extended R-automata

Now we are ready to show the translation of the timed automaton into an extended R-automaton. Intuitively, we equip the region graph induced by a given timed automaton with counters whose values are updated as we move along a path in the region graph. These counters keep the information about the minimal distances between the fractional parts of the clocks. The distance is not characterized in an absolute manner, but relatively to a sampling unit ε . Let the counter values be obtained after following a path in the region graph. The counters say how many ε 's do there have to be at least between the fractional parts of two clocks in any state reachable by a concrete run of the timed automaton along this path in the region graph.

This section is organized as follows. First, we describe how to translate a timed automaton with at most one clock reset along each transition into an extended R-automaton. Then we show two technical properties of the constructed extended R-automaton (Lemma 9 and Lemma 10). In the rest of this section we prove the correspondence between the counter values along runs of the extended R-automaton and the minimal distances between the fractional parts of the clock values in the timed automaton (Lemma 11 and Lemma 12).

Construction.

Let G be the region graph induced by a given timed automaton A' with at most one reset in each transition. We build an extended R-automaton R from this region graph G . The extended R-automaton R has a state corresponding to each node in the region graph G and three auxiliary states for each edge in the region graph G corresponding to a discrete transition (an edge labeled by $a \in \Sigma$). The initial state is the state corresponding to the node $\langle s_0, \{v_0\} \rangle$. The accepting states are the states corresponding to the nodes $\langle s, D \rangle$, where $s \in F$. We introduce two counters C_{xy}, C_{yx} for each pair of clocks $x, y \in \mathcal{C}$. We use only the P values from the extended R-automaton and in the following we will refer to them simply by C_{xy}, C_{yx} .

Since encoding of a single edge might need to perform multiple counter updates, we introduce a sequence of four transitions and three auxiliary states between them for each edge in G corresponding to a discrete transition of the timed automaton A' . These transitions are labeled by the same letter as the original edge. More precisely, let us have an edge in G from $\langle s', D' \rangle$ to $\langle s, D \rangle$ labeled by a , where $a \in \Sigma$. Then we create three auxiliary states s_1, s_2, s_3 (these states are unique for this transition, formally we should write $s_1^{(s,D,a,s',D')}$, $s_2^{(s,D,a,s',D')}$, $s_3^{(s,D,a,s',D')}$, but without confusion, we skip the superscript) and four transitions from $\langle s', D' \rangle$ to s_1 , from s_1 to s_2 , from s_2 to s_3 , and from s_3 to $\langle s, D \rangle$, all of them labeled by a .

For edges corresponding to a time pass transition in A (edges labeled by δ), we introduce only one transition directly leading to the state corresponding to the target node labeled by δ . More precisely, let us have an edge from $\langle s', D' \rangle$ to $\langle s, D \rangle$ labeled by δ in G . We create a transition in R from $\langle s', D' \rangle$ to $\langle s, D \rangle$

labeled by δ . Later on, we show how to get rid of these transitions (and of the letter δ).

Now we show how to label the transitions by effects. The transitions labeled by δ and the transitions corresponding to an edge in G from $\langle s', D' \rangle$ to $\langle s, D \rangle$ labeled by a , $a \in \Sigma$, where either $D = D'$ (no clock is reset) or a clock x is reset such that $x =_{D'} 0$ (the clock had zero fractional part before reset) are labeled by the effect $(0, \dots, 0)$ (all counters are left unchanged).

In other cases, we have transitions corresponding to an edge in G from $\langle s', D' \rangle$ to $\langle s, D \rangle$ labeled by a where a clock with non-zero fractional part is reset. Let us denote this clock by x . These transitions are labeled by effects created according to the following four cases. Counters which are not mentioned are left unchanged (the instruction is 0 on all four transitions). The instructions are denoted by pairs $C : e_1, e_2, e_3, e_4$, where C is the counter, to which the instructions are applied and e_1, e_2, e_3, e_4 are the instructions (e_i is a part of the effect on the i -th transition).

1. The region D' has a clock a with zero fractional part (depicted in Figure III.7).
 - $C_{ax}, C_{xa} : r(\emptyset), 0, 0, 0$
 - $\forall u \neq a, x. C_{ux} : *C_{ua}, 0, 0, 0$ and $C_{xu} : *C_{au}, 0, 0, 0$.

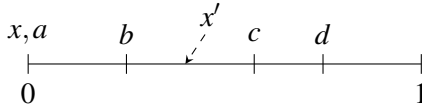


Figure III.7: The region D' has a clock a with zero fractional part. The letter x' denotes the position of the clock x in D' (before it was reset).

2. The region D' has clocks a, d such that the fractional part of a is smaller than or equal to the fractional part of x and the fractional part of d is greater than or equal to the fractional part of x (depicted in Figure III.8).
 - $\forall u \neq a, x. C_{xu} : *C_{au}, 1, 1, 0$
 - $\forall u \neq d, x. C_{ux} : *C_{ud}, 1, 1, 0$
 - $C_{xa} : 0, r(\{C_{da}, C_{xu} | \forall u \neq a, x\}), 1, 1$
 - $C_{dx} : 0, r(\{C_{da}, C_{ux} | \forall u \neq d, x\}), 1, 1$

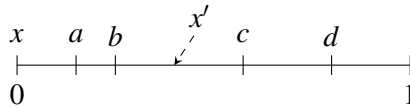


Figure III.8: The region D' has no clock with zero fractional part and the fractional part of x is neither strictly smaller nor strictly greater than all other clocks. The letter x' denotes the position of the clock x in D' (before it was reset).

3. The clock x has strictly smaller fractional part than other clocks in D' (depicted in Figure III.9). We denote a clock with the smallest fractional part greater than the fractional part of x by a and a clock with the greatest fractional part by d .

- $u \neq x. C_{xu} : 1, 0, 0, 0$
- $\forall u \neq d, x. C_{ux} : *C_{ud}, 1, 1, 0$
- $C_{dx} : 0, r(\{C_{da}, C_{ux} | \forall u \neq d, x\}), 1, 1$

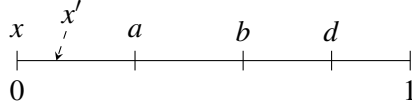


Figure III.9: The clock x has the smallest (strictly) fractional part in the region D' . The letter x' denotes the position of the clock x in D' (before it was reset).

4. The clock x has strictly greater fractional part than other clocks in D' (depicted in Figure III.10). We denote a clock with the greatest fractional part smaller than the fractional part of x in D' by d and a clock with the smallest fractional part in D' by a .

- $u \neq x. C_{ux} : 1, 0, 0, 0$
- $\forall u \neq a, x. C_{xu} : *C_{au}, 1, 1, 0$
- $C_{xa} : 0, r(\{C_{da}, C_{xu} | \forall u \neq a, x\}), 1, 1$

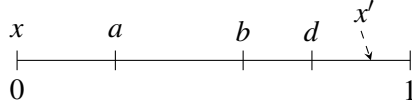


Figure III.10: The clock x has the greatest (strictly) fractional part in the region D' . The letter x' denotes the position of the clock x in D' (before it was reset).

The informal alternative description of the updates by effects is that a counter is incremented if the distance between the two corresponding clocks grows and a counter is reset to 2 if the distance between the two corresponding clocks decreases and then the counters are updated to satisfy $D \models \overline{xy}z \Rightarrow C_{xy} + C_{yz} \leq C_{xz}$. Since we use the max function to maintain this property, we need to take care of the preorder \lesssim . In order to do this, we need all the copy instructions in Items 2 – 4. The important property of \lesssim is formalized in the following lemma. The proof is rather technical and analyzes the items in the construction and the semantics of extended R-automata. We call the states of R which are not auxiliary *complete*.

Lemma 9 *For all reachable complete states $\langle \langle q, D \rangle, \vec{C}, \lesssim \rangle$ of R , the following holds:*

- (i) $C_{bc} \lesssim C_{ad}$ if and only if for all $v \in D$, $\overline{bc}_v < \overline{ad}_v$, and
- (ii) $C_{ab} \simeq C_{cd}$ if and only if for all $v \in D$, $\overline{ab}_v = \overline{cd}_v > 0$.

Proof. We show by induction on the length of the shortest path reaching $\langle \langle q, D \rangle, \vec{C}, \lesssim \rangle$ that the claim holds. The basic step is trivial. For the induction step, observe that the claim that for all $v \in D$, $\overline{bc}_v < \overline{ad}_v$ is equivalent to $(D \models \overline{abc} \wedge D \models \overline{cda}) \vee (a =_D b \wedge D \models \overline{acd}) \vee (c =_D d \wedge D \models \overline{abc})$ and the claim that for all $v \in D$, $\overline{ab}_v = \overline{cd}_v > 0$ is equivalent to $a =_D c \wedge b =_D d$.

Point (i), " \Rightarrow ": Correctness of all inequalities introduced by Item 1 of the construction follows from IH.

Item 2 of the construction introduces inequalities $C_{xu} \lesssim C_{au}$ and $C_{ux} \lesssim C_{ud}$, because of the copy instruction (Point 3 in the semantics introduces equality) and then C_{xu}, C_{ux} are incremented by the instruction 1, which breaks the equality into inequality (Point 1 in the semantics). But it is clear from the analysis of the region D' and the observations above that the claim is satisfied. Item 2 also introduces inequalities by resets. The reset instructions are delayed by one transition (they take place on the second transition in the sequence) and therefore the inequalities $C_{xu} \lesssim C_{au}$ and $C_{ux} \lesssim C_{ud}$ are already established. This prevents the inequalities $C_{xa} \lesssim C_{au}, C_{dx} \lesssim C_{ud}$ to appear in the preorder. It is easy to verify from the region that the remaining inequalities which are established satisfy the claim. It follows from IH that the inequalities introduced by the transitive closure satisfy the claim.

Item 3 does not introduce any new inequalities for C_{xu} , because there is no other counter C_{ab} such that $C_{ab} \simeq C_{xu}$ (IH, Point (ii)). The argument for the inequalities created by copying and resets is the same as for the previous item.

Item 4 is dual to the previous item.

Point (i), " \Leftarrow ": The fact that all required inequalities are created by Item 1 follows from IH.

Items 2 – 4 have to create new inequalities for counters containing the clock x (we can find all of them by inspecting the regions). The copy instructions put $C_{xu} \simeq C_{au}$ and $C_{ux} \simeq C_{ud}$ (Point 3 in the semantics). The counters C_{xu}, C_{ux} are then incremented by the instruction 1, while the counters C_{au}, C_{ud} stay unchanged (instruction 0). This results in the inequalities $C_{xu} \lesssim C_{au}$ and $C_{ux} \lesssim C_{ud}$. The clocks C_{xa}, C_{dx} are reset by an instruction which contains all the important clocks. This (as defined in Point 2 of the semantics, together with the transitive closure) creates all the necessary inequalities.

Point (ii), " \Rightarrow ": Item 1 creates equalities by the copy instruction (Point 3) and the transitive closure, but the correctness follows immediately from the fact that $a =_D x$ and from IH (for the transitive closure).

Items 2 – 4 introduce equalities by the copy instructions and the transitive closure, but because the clocks C_{xu}, C_{ux} are incremented by 1 and the clocks C_{au}, C_{ud} are left unchanged, the equalities introduced by the copy instructions are broken. The equalities introduced by the transitive closure satisfy the claim (IH).

Point (ii), " \Leftarrow ": New equalities required by the region in Item 1 are created by the copy instructions and the transitive closure. The other required equalities follow from IH. Note that $\overline{ax}_v = \overline{xa}_v = 0$ for all $v \in D$ and therefore the equality $C_{ax} \simeq C_{xa}$ is not required.

Items 2 – 4 do not move any two clocks together and therefore the claim holds from IH. ■

The the following lemma formulates (most of) the essential properties of the construction we need for the proof of the correctness of the reduction. Because of this lemma, we do not have to refer to \lesssim and max operations anymore.

Lemma 10 *For all reachable complete states $\langle\langle q, D \rangle, \bar{C}, \lesssim\rangle$ of R , the following holds:*

- (i) *if $D \models \overline{xy}$ then $C_{xy} + C_{yz} \leq C_{xz}$,*
- (ii) *if $x =_D y$ then $C_{xy} = 0$ and $C_{xu} = C_{yu}$, $C_{ux} = C_{uy}$ for all clocks u .*
- (iii) *if $x \neq_D y$ then $C_{xy} \geq 2$, $C_{yx} \geq 2$.*

Proof. Point (i) follows directly from Lemma 9 and Step 4 in the definition of the semantics of extended R-automata.

The first part of Point (ii) follows from Item 1 in the construction of R and the fact that this counter can be changed only along a transition which leads to a state $\langle\langle q', D' \rangle, \bar{C}', \lesssim'\rangle$, where $x \neq_{D'} y$ (follows straightforwardly from the construction). The second part follows from Lemma 9 and an observation that counters equivalent with respect to \lesssim contain the same values.

Point (iii) follows from a simple inductive argument. If $x \neq_D y$ holds and it did not hold in the previous state then C_{xy}, C_{yx} is either updated by a copy from a counter with value greater than or equal to 2 (Item 1) or by a copy or reset followed by two increments (Item 2). Especially, Items 3 and 4 cannot be applied. If $x \neq_D y$ holds and it held also in the previous state then C_{xy}, C_{yx} is either incremented (Items 3 and 4) or updated by a copy from a counter with value greater than or equal to 2 (Item 1) or by a copy or reset followed by two increments (Items 2, 3, and 4). ■

The property formalized in Point (i) of the previous lemma is the reason for extending the R-automata with the max operations. The preorder \lesssim is a technical construction thanks to which we are able to reduce limitedness for R-automata with max operations to limitedness of R-automata.

Correspondence between A' and R .

Now we formulate correspondence properties between the timed automaton A' and the extended R-automaton R constructed as above. Let us recall that we ignore N and M values of the counters and denote the P values by C . For instance, a state $\langle\langle q, D \rangle, (\bar{N}, \bar{M}, \bar{P}), \lesssim\rangle$ is written as $\langle\langle q, D \rangle, \bar{C}, \lesssim\rangle$. Let for a state in a run of an extended R-automaton, the value of the state be the maximal counter value in this state (the P -value). Let for a run ρ of an extended R-automaton, the maximum counter value along this run be the maximal state value along this run. This is the value $P\{\rho\}$, but to avoid confusion, we denote it by $\max\{\rho\}$ here.

Let us say that a valuation $v \in D$ for some ε satisfies the counter valuation \bar{C} (denoted by $v \models_\varepsilon \bar{C}$) if for each pair of clocks x, y , $\overline{xy}_v / \varepsilon \geq C_{xy}$ (or equivalently, $\overline{xy}_v \geq C_{xy} \cdot \varepsilon$).

Lemma 11 *Let R be the extended R-automaton constructed from the region graph G induced by a timed automaton A . Let $\rho = \langle\langle q_0, \{v_0\}\rangle, \bar{C}_0, \emptyset\rangle \longrightarrow \langle\langle q, D \rangle, \bar{C}, \lesssim\rangle$ be a run in R , $\sigma = \langle q_0, \{v_0\}\rangle \longrightarrow \langle q, D \rangle$ be the corresponding path in G , and $\varepsilon = 1/(2 \cdot \max\{\rho\})$. For all $v \in D_\varepsilon$ such that $v \models_\varepsilon \bar{C}$ there is a run ρ' in $\llbracket A \rrbracket_\varepsilon$ ending in (q, v) such that $\rho' \models \sigma$. Also, there is a $v \in D_\varepsilon$ such that $v \models_\varepsilon \bar{C}$.*

Proof. By induction on the length of σ . The basic step is trivial.

For the induction step, let us first observe that the maximum counter value along ρ is greater than or equal to the maximum counter value along its prefixes. Let $v \in D_\varepsilon$, $v \models_\varepsilon \bar{C}$. We have to find $v' \in D'_\varepsilon$, $v' \models_\varepsilon \bar{C}'$, where $\langle \langle q', D' \rangle, \bar{C}', \lesssim' \rangle$ is the previous complete state of ρ , such that v can be reached from v' along the edge from $\langle q', D' \rangle$ to $\langle q, D \rangle$. We discuss different types of this edge.

We first discuss the case where the edge leads to the immediate time successor. Let x be a clock with minimal fractional part in v . If $\text{fr}(v(x)) > 0$ then $v'(y) = v(y) - v(x)$ for all $y \in \mathcal{C}$. If $\text{fr}(v(x)) = 0$ then $v'(y) = v(y) - \varepsilon$ for all $y \in \mathcal{C}$. Because the minimal distance between two clocks is $2 \cdot \varepsilon$ (follows from IH and Lemma 10), $v' \in D'_\varepsilon$ in both cases. Also, $v' \models \bar{C}'$, because $C' = C$ (instructions on all counters are 0) and the differences between the clocks do not change.

We discuss an edge along which a clock (denote x) is reset. The case where $\text{fr}(v'(x)) = 0$ (x has zero fractional part in D' , $x =_{D'} 0$) clearly holds, because neither distances between the fractional parts of the clocks nor the counters change. For the other case, we discuss different types of the regions D, D' corresponding to the cases in the construction of R separately. Let i be the integral part of the clock x in D' . If there is a clock z such that $\bar{x}z_{v'} = 0$ then $v'(y) = v(y)$ for all $y \neq x$ and $v'(x) = i + \text{fr}(v(z))$. Otherwise, except for one case where it is mentioned explicitly, $v'(y) = v(y)$ for all $y \neq x$ and $v'(x) = i + \max\{\text{fr}(v(z) + C'_{zx} \cdot \varepsilon) \mid z \neq x\}$. The construction of the valuation v' for x is depicted in Figure III.11.

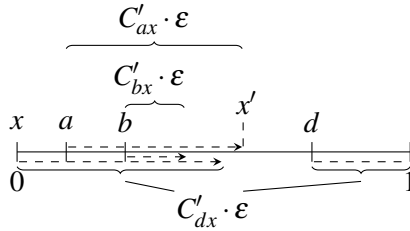


Figure III.11: Illustration of the calculation of the value of x in the valuation v' . The positions of the clocks correspond to the valuation v , where $v(x) = 0$ (x was reset), $v(a) = 0.12$, $v(b) = 0.24$, $v(d) = 0.75$. The values of the counters are $C_{ax} = 10$, $C_{bx} = 3$, $C_{dx} = 17$. The sampling rate is $\varepsilon = 0.04$ and thus $C_{ax} \cdot \varepsilon = 0.4$, $C_{bx} \cdot \varepsilon = 0.12$, $C_{dx} \cdot \varepsilon = 0.68$. Then $v'(x)$, depicted by x' , is $\max\{0.52, 0.36, 0.43\} = 0.52$.

As the first case we consider the situation where the region D' has a clock a with zero fractional part (depicted in Figure III.7, Item 1 in the construction). We denote a clock with the greatest fractional part smaller than the fractional part of x by b (there is always one such clock, since b could be the clock a). If it exists, then we also denote a clock with the smallest fractional part greater than the fractional part of x by c .

We have to show that $v' \in D'_\varepsilon$ and that $v' \models_\varepsilon \bar{C}'$. First we show that $v' \in D'_\varepsilon$. If there is a clock y such that $x =_{D'} y$ then clearly $v' \in D'_\varepsilon$. Otherwise, we have

to show that $\text{fr}(v'(b)) < \text{fr}(v'(x))$ and if c exists then also that $\text{fr}(v'(x)) < \text{fr}(v'(c))$. To show that $\text{fr}(v'(b)) < \text{fr}(v'(x))$, we need to show that $\text{fr}(v(b) + C'_{bx} \cdot \varepsilon) = \text{fr}(v(b)) + C'_{bx} \cdot \varepsilon$ and then the rest follows from the construction of v' . Since $C_{ba} = C'_{ba} \geq C'_{bx} + C'_{xa}$ and $C'_{xa} \geq 2$ (Lemma 10), we have that $C_{ba} > C'_{bx}$ and from the fact that $v \models_{\varepsilon} \bar{C}$ we have that $\bar{b}a_v > C'_{bx} \cdot \varepsilon$ and thus $\text{fr}(v(b)) + C'_{bx} \cdot \varepsilon < 1$. To show that $\text{fr}(v'(x)) < \text{fr}(v'(c))$, we discuss the following two cases. Let us denote the clock chosen by the max function in the construction of the value $v'(x)$ by z .

- If the clock z does not have the same fractional part as c in D' then we have that $C_{zc} = C'_{zc} \geq C'_{zx} + C'_{xc}$ and $C'_{xc} \geq 2$ (Lemma 10), thus $C_{zc} > C'_{zx}$. From the fact that $v \models_{\varepsilon} \bar{C}$ and from the construction of v' we have that $\bar{z}c_{v'} \geq C'_{zc} \cdot \varepsilon$, $\bar{z}x_{v'} = C'_{zx} \cdot \varepsilon$. This gives that $\bar{z}c_{v'} > \bar{z}x_{v'}$ and together with the facts that $\text{fr}(v'(b)) < \text{fr}(v'(x))$ and $\text{fr}(v'(b)) < \text{fr}(v'(c))$ (the shape of the region) we obtain the required inequality $\text{fr}(v'(x)) < \text{fr}(v'(c))$.
- If the clock z has the same fractional part as c in D' then it suffices to observe that $C'_{cx} < k$ and thus $C'_{cx} \cdot \varepsilon < 1$.

Now we show that $v' \models_{\varepsilon} \bar{C}'$. If there is a clock y such that $x =_{D'} y$ then the fact that $v' \models_{\varepsilon} \bar{C}'$ follows directly from Lemma 10. Otherwise, we have to check all the counters. For all counters C'_{uv} such that $u, v \neq x$, $C'_{uv} = C_{uv}$ and from the construction of v' , $\bar{u}v_{v'} \geq C'_{uv} \cdot \varepsilon$. For counters C'_{ux} (for all clocks u), the fact that $\bar{u}x_{v'} \geq C'_{ux} \cdot \varepsilon$ follows directly from the construction of v' . For the counters C'_{xu} we consider two cases. Let us denote the clock chosen by the max function in the construction of the value $v'(x)$ by z .

- If the clock z does not have the same fractional part as u in D' then we have again two possibilities.
 - If $D' \models \bar{x}u\bar{z}$ then we have that $C_{zu} = C'_{zu} \geq C'_{zx} + C'_{xu}$ (Lemma 10). From the fact that $v \models_{\varepsilon} \bar{C}$ and from the construction of v' we have that $\bar{z}u_{v'} \geq C'_{zu} \cdot \varepsilon$, $\bar{z}x_{v'} = C'_{zx} \cdot \varepsilon$ and therefore $\bar{x}u_{v'} = \bar{z}u_{v'} - \bar{z}x_{v'} > C'_{xu} \cdot \varepsilon$.
 - If $D' \models \bar{x}z\bar{u}$ then we have that $\bar{x}u_{v'} > \bar{x}z_{v'}$. From the construction of v' we have that $\bar{x}z_{v'} = 1 - (C'_{zx} \cdot \varepsilon)$ and from the construction of ε we have that $1 \geq 2 \cdot \max\{C'_{zx}, C'_{xu}\} \cdot \varepsilon$. This together gives that $\bar{x}z_{v'} \geq (C'_{xu} \cdot \varepsilon)$.
- If the clock z has the same fractional part as u in D' then it suffices to observe that $C'_{xu} + C'_{ux} \leq k$ and thus $1 \geq (C'_{xu} + C'_{ux}) \cdot \varepsilon$. From the construction of v' we have that $\bar{x}u_{v'} = 1 - (C'_{ux} \cdot \varepsilon)$ and thus $\bar{x}u_{v'} \geq C'_{xu} \cdot \varepsilon$.

As the second case we consider the situation where Item 2 in the construction applies. There, the region D' has clocks a, d such that the fractional part of the clock a is smaller than or equal to the fractional part of x and the fractional part of the clock d is greater than or equal to the fractional part of x (depicted in Figure III.8). We denote a clock with the greatest fractional part smaller than the fractional part of x by b (there is always one such clock, since b could be the clock a). We also denote a clock with the smallest fractional part greater than the fractional part of x by c (there is always one such clock, since b could be the clock d).

The argument for case is the same as for the first case, with a simplification that there is always a clock c .

As the third case we consider the situation where Item 3 in the construction applies. There, x has strictly smaller fractional part than other clocks in the region (depicted in Figure III.9). We denote a clock with the smallest fractional part greater than the fractional part of x by a (there is always one such clock, since $|\mathcal{C}| \geq 2$).

For this case, we need to define v' in a dual way. Let i be the integral part of the clock x in D' . Let $v'(y) = v(y), y \neq x$ and $v'(x) = i + \min\{\text{fr}(v(z)) - C'_{xz} \cdot \varepsilon \mid z \neq x\}$.

We have to show that $v' \in D'_\varepsilon$ and that $v' \models_\varepsilon \bar{C}'$. First we show that $v' \in D'_\varepsilon$. We have to show that $(\text{fr}(v(z)) - C'_{xz} \cdot \varepsilon) > 0$ for all clocks z and that $\text{fr}(v'(x)) < \text{fr}(v'(a))$. The first part follows from the fact that $\bar{x}z_v \geq C_{xz} \cdot \varepsilon$, $v(z) = v'(z)$, and $C'_{xz} < C_{xz}$. At this place, we use the fact that the value of C_{xz} is incremented along these transitions in the extended R-automaton construction. The second fact follows from the first one and from the fact that $C'_{xa} \geq 2$ (Lemma 10).

Now we show that $v' \models_\varepsilon \bar{C}'$. The argument is 'dual' to the argument for the first case. For all counters C'_{uv} such that $u, v \neq x$, $C'_{uv} \leq C_{uv}$ and from the construction of v' , $\bar{u}v_{v'} \geq C'_{uv} \cdot \varepsilon$. For counters C'_{xu} (for all clocks u), the fact that $\bar{x}u_{v'} \geq C'_{xu} \cdot \varepsilon$ follows directly from the construction of v' . For the counters C'_{ux} we consider two cases. Let us denote the clock chosen by the min function in the construction of the value $v'(x)$ by z .

- If the clock z does not have the same fractional part as u in D' then we have again two possibilities.
 - If $D' \models \bar{x}z_u$ then we have that $C_{uz} = C'_{uz} \geq C'_{ux} + C'_{xz}$ (Lemma 10). From the fact that $v' \models_\varepsilon \bar{C}$ and from the construction of v' we have that $\bar{u}z_{v'} \geq C'_{uz} \cdot \varepsilon$, $\bar{x}z_{v'} = C'_{xz} \cdot \varepsilon$ and therefore $\bar{u}x_{v'} = \bar{u}z_{v'} - \bar{x}z_{v'} > C'_{ux} \cdot \varepsilon$.
 - If $D' \models \bar{x}u_z$ then we have that $\bar{u}x_{v'} > \bar{x}z_{v'}$. From the construction of v' we have that $\bar{x}z_{v'} = 1 - (C'_{xz} \cdot \varepsilon)$ and from the construction of ε we have that $1 \geq 2 \cdot \max\{C'_{xz}, C'_{ux}\} \cdot \varepsilon$. This together gives that $\bar{x}z_{v'} \geq (C'_{ux} \cdot \varepsilon)$.
- If the clock z has the same fractional part as u in D' then it suffices to observe that $C'_{xu} + C'_{ux} \leq k$ and thus $1 \geq (C'_{xu} + C'_{ux}) \cdot \varepsilon$. From the construction of v' we have that $\bar{u}x_{v'} = 1 - (C'_{xu} \cdot \varepsilon)$ and thus $\bar{u}x_{v'} \geq C'_{ux} \cdot \varepsilon$.

As the fourth case we consider the situation where Item 4 in the construction applies. There, x has strictly greater fractional part than other clocks in the region (depicted in Figure III.10). We denote a clock with the greatest fractional part smaller than the fractional part of x in D' by d (there is always one such clock, since $|\mathcal{C}| \geq 2$).

We construct v' as for the first case. The correctness argument is 'dual' to the argument from the third case.

We have to show that $v' \in D'_\varepsilon$ and that $v' \models_\varepsilon \bar{C}'$. First we show that $v' \in D'_\varepsilon$. We have to show that $(\text{fr}(v(z)) + C'_{zx} \cdot \varepsilon) < 1$ for all clocks z and that $\text{fr}(v'(d)) < \text{fr}(v'(x))$. The first part follows from the fact that $\bar{x}z_v \geq C_{zx}$, $v(z) = v'(z)$, and $C'_{zx} < C_{zx}$. At this place, we use the fact that the value of C_{zx} is incremented along these transitions in the extended R-automaton construction.

The second fact follows from the first one and from the fact that $C'_{dx} \geq 2$ (Lemma 10).

Now we have to show that $v' \models_{\varepsilon} \bar{C}'$. The argument is the same as the argument for the first case, with the difference that for the counters C'_{uv} such that $u, v \neq x$, we have that $C'_{uv} \leq C_{uv}$.

It remains to show that there is a valuation $v \in D_{\varepsilon}$ such that $v \models_{\varepsilon} \bar{C}$. We construct v in the following way. Let the integral parts of all clocks correspond to D . Let us order the clocks $c_1 \leq c_2 \leq \dots \leq c_{|\mathcal{C}|}$ in such a way that $c_i \leq c_j$ if and only if $\text{fr}(\bar{v}(c_i)) \leq \text{fr}(\bar{v}(c_j))$ for all $\bar{v} \in D$. If c_1 has zero fractional part in D then $\text{fr}(v(c_1)) = 0$, otherwise, $\text{fr}(v(c_1)) = \varepsilon$. For all $i > 1$, let $\text{fr}(v(c_i)) = C_{c_1 c_i} \cdot \varepsilon$. We need to show that $v \in D_{\varepsilon}$ and that $v \models_{\varepsilon} \bar{C}$. The fact that $v \in D_{\varepsilon}$ follows directly from Lemma 10 and from the construction of ε .

We need to show that $v \models_{\varepsilon} \bar{C}$. For all $i < j$, $\bar{c}_i \bar{c}_j v \geq C_{c_i c_j} \cdot \varepsilon$, because of the construction of v and the fact that $C_{c_1 c_i} + C_{c_i c_j} \leq C_{c_1 c_j}$ (Lemma 10). We also know that $\bar{c}_i \bar{c}_1 v \geq C_{c_1 c_i} \cdot \varepsilon$ for all i , because of the construction of ε . For all other $i > j$, $\bar{c}_i \bar{c}_j v \geq C_{c_i c_j} \cdot \varepsilon$, because $\bar{c}_i \bar{c}_j v > 1 - (C_{c_1 c_i} \cdot \varepsilon)$ and $1 \geq (C_{c_1 c_i} + C_{c_i c_j}) \cdot \varepsilon$. ■

We also prove that the maximum counter value of a path constraints the ε from above.

Lemma 12 *Let R be the extended R -automaton constructed from the region graph G induced by a timed automaton A . Let $\rho = \langle \langle q_0, \{v_0\} \rangle, \bar{C}_0, \emptyset \rangle \longrightarrow \langle \langle q, D \rangle, \bar{C}, \lesssim \rangle$ be a run in R , $\sigma = \langle q_0, \{v_0\} \rangle \longrightarrow \langle q, D \rangle$ be the corresponding path in G and $\rho' = \langle q_0, v_0 \rangle \longrightarrow_{\varepsilon} \langle q, v \rangle$ be a run in $\llbracket A \rrbracket_{\varepsilon}$ for some ε such that $\rho' \models \sigma$. Then for all pairs of clocks x, y , $\bar{x}y_v \geq (C_{xy}/2) \cdot \varepsilon$.*

Proof. By induction on the length of σ . The basic step is trivial. For the induction step, we show that if the runs of R and A end in the states $\langle \langle q', D' \rangle, \bar{C}', \lesssim' \rangle$ and $\langle q', v' \rangle$, respectively, satisfying the condition, i.e., for all pairs of clocks x, y , $\bar{x}y_{v'} \geq (C_{xy}/2) \cdot \varepsilon$, then the condition is also satisfied after transitions leading to the next (complete) states $\langle \langle q, D \rangle, \bar{C}, \lesssim \rangle$ and $\langle q, v \rangle$. We discuss the types of transitions.

We first discuss the case where the edge leads to the immediate time successor. The condition is clearly satisfied, because neither the differences between the clocks nor the counter values change after a time transition.

We discuss an edge along which a clock (denote x) is reset. The case where $\text{fr}(v'(x)) = 0$ (x has zero fractional part in D' , $x =_{D'} 0$) clearly keeps the condition satisfied, because neither the differences between the clocks nor the counter values change after reset of x . For the other case, we discuss several different types of the regions D, D' .

As the first case we consider the situation where the region D' has a clock a with zero fractional part (depicted in Figure III.7, Item 1 in the construction). For the clocks $u, v \neq x$, the distances between the fractional parts do not change and $C_{uv} = C'_{uv}$, $C_{vu} = C'_{vu}$. For each clock u , $C_{xu} = C'_{au}$, $C_{ux} = C'_{ua}$, hence the condition is satisfied from IH.

As the second case we consider the situation where Item 2 in the construction applies. There, the region D' has clocks a, d such that the fractional part

of the clock a is smaller than or equal to the fractional part of x and the fractional part of the clock d is greater than or equal to the fractional part of x (depicted in Figure III.8). We denote a clock with the greatest fractional part smaller than the fractional part of x by b (there is always one such clock, since b could be the clock a). We also denote a clock with the smallest fractional part greater than the fractional part of x by c (there is always one such clock, since b could be the clock d).

First, we look at the distances $\overline{xa}, \overline{dx}, \overline{da}$. We have that $C_{xa} = C_{dx} = 2$, but already from the region we know that $\overline{xa}_v \geq \varepsilon, \overline{dx}_v \geq \varepsilon$. Lemma 10 gives us that $C_{da} = \max\{C'_{da}, C_{xa} + C_{dx} = 4\}$, so the condition either holds from IH ($C'_{da} > 4$) or because $\overline{xa}_v \geq 2 \cdot \varepsilon$ (from the region).

For the distances between the clocks a and d (avoiding x in D), neither distances nor the counter values change.

For the distances between the clocks c and b such that $\text{fr}(v'(c)) > \text{fr}(v'(b))$ (equivalently, $c >_D b, c >_{D'} b$) we have to analyze the counters carefully. (This is the case where we pass through x in D when going from c to b ; in the following argumentation we assume that c is different from a and b is different from d , but it is easy to see that the same arguments, even a bit simplified, would work if this assumption does not hold.) If $C_{cb} = C'_{cb}$ then the validity of the condition holds from IH. But since the counter C_{da} can be bigger than C'_{da} , the counter C_{cb} might be bigger than C'_{cb} and we have to show that the distance between c and b is big enough even for C_{cb} (it is the same in both v' and v). If $C_{cb} > C'_{cb}$ then we know that $C'_{da} < 4$ and $C'_{cb} = C'_{cd} + C'_{da} + C'_{ab}$ (this follows from Lemma 10). But then we know that $C_{cb} = C_{cd} + C_{da} + C_{ab} = C'_{cd} + 4 + C'_{ab}$. We also have that $cb_v = \overline{cd}_v + \overline{da}_v + \overline{ab}_v$. From IH we know that $\overline{cd}_v \geq (C_{cd}/2) \cdot \varepsilon, \overline{ab}_v \geq (C_{ab}/2) \cdot \varepsilon$, from the region we have that $\overline{da}_v \geq 2 \cdot \varepsilon$. Together, $cb_v \geq (C_{cb}/2) \cdot \varepsilon$.

Now we look at the distances between x and other clocks in the region b such that $a <_D b$. Directly from the construction of R we have that $C_{xb} = 2 + C_{ab} = 2 + C'_{ab}$. From IH we know that $\overline{ab}_v \geq (C_{ab}/2) \cdot \varepsilon$, from the region we have that $\overline{xa}_v \geq \varepsilon$. Together, $\overline{xb}_v \geq (C_{xb}/2) \cdot \varepsilon$.

It remains to check the distances between clocks b such that $b <_D d$ and x . This case is symmetrical to the previous case.

As the third case we consider the situation where Item 3 in the construction applies. There, x has strictly smaller fractional part than other clocks in the region (depicted in Figure III.9). We denote a clock with the smallest fractional part greater than the fractional part of x by a (there is always one such clock, since $|\mathcal{C}| \geq 2$). We also denote a clock with the greatest fractional part in D' by d (there is always one such clock, since $|\mathcal{C}| \geq 2$).

First, we look at the distances between x and other clocks in the region b . From the construction we have that $C_{xb} = C'_{xb} + 1$. From IH we know that $\overline{xb}_{v'} \geq (C'_{xb}/2) \cdot \varepsilon$, from the region we have that $\overline{xb}_v \geq \overline{xb}_{v'} + \varepsilon$. Together, $\overline{xb}_v \geq (C_{xb}/2) \cdot \varepsilon$.

Now we check the distances between clocks b and x . $C_{dx} = 2$, but already from the region we know that $\overline{dx}_v \geq \varepsilon$. For the other clocks we have directly

from the construction of R that $C_{bx} = C_{bd} + 2 = C'_{bd} + 2$. From IH we know that $\overline{bd}_v \geq (C_{bd}/2) \cdot \varepsilon$, from the region we have that $\overline{da}_v \geq \varepsilon$. Together, $\overline{bx}_v \geq (C_{bx}/2) \cdot \varepsilon$.

For the distances between the clocks c and b such that $\text{fr}(v'(c)) > \text{fr}(v'(b))$ (equivalently, $c >_D b$, $c >_{D'} b$) we have to analyze the counters carefully. (In the following argumentation we assume that c is different from a and b is different from d , but it is easy to see that the same arguments, even a bit simplified, would work if this assumption does not hold.) If $C_{cb} = C'_{cb}$ then the validity of the condition holds from IH. If $C_{cb} > C'_{cb}$ then we know that $C'_{dx} = 2$ and $C'_{cb} = C'_{cd} + C'_{dx} + C'_{xb}$ (this follows from Lemma 10). But then we know that $C_{cb} = C_{cd} + C_{dx} + C_{xb} = C'_{cd} + 2 + C_{xb}$. We also have that $\overline{cb}_v = \overline{cd}_v + \overline{dx}_v + \overline{xb}_v$. From IH we know that $\overline{cd}_v \geq (C_{cd}/2) \cdot \varepsilon$, we have shown that $\overline{xb}_v \geq (C_{xb}/2) \cdot \varepsilon$, from the region we have that $\overline{dx}_v \geq \varepsilon$. Together, $\overline{cb}_v \geq (C_{cb}/2) \cdot \varepsilon$.

For the distances between the clocks a and d (avoiding x in D, D'), neither distances nor the counter values change.

As the fourth case we consider the situation where Item 4 in the construction applies. This case is dual to the third case. ■

6 Decidability Proof

First we show that Theorem 1 is true for timed automata with one clock.

Lemma 13 *For a given timed automaton A with the set of clocks \mathcal{C} such that $|\mathcal{C}| = 1$, $L_{1/2}(A) = L(A)$ and $L_{1/2}^\omega(A) = L^\omega(A)$.*

Proof. Let us denote the clock by x . For each run over w in $\llbracket A \rrbracket_{\mathbb{R}_0^+}$, we construct a run in $\llbracket A \rrbracket_{1/2}$ as follows. We modify the time delays so that all discrete transitions taken with $\text{int}(x) = i$ and $\text{fr}(x) \neq 0$ are now taken with $\text{int}(x) = i$ and $\text{fr}(x) = 1/2$. Clearly, there is such a run in $\llbracket A \rrbracket_{\mathbb{R}_0^+}$, because for all $i \in \mathbb{N}$, all valuations with $\text{int}(x) = i$ and $\text{fr}(x) \neq 0$ are untimed bisimilar. Also, such a run is also a run in $\llbracket A \rrbracket_{1/2}$. ■

For the other cases, we first show how to transform a given timed automaton into a timed automaton which resets at most one clock along each transition and which is equivalent with respect to the sampling problem. For each discrete transition labeled by a with a guard g and resetting $Y \subseteq \mathcal{C}$, we create a sequence of $|\mathcal{C}|$ transitions (and $|\mathcal{C}| - 1$ auxiliary non-accepting states between them) labeled by a . These transitions reset clocks from Y one by one (let us if $Y \neq \emptyset$ denote the first reset clock by x). The first transition is guarded by g and the guards on the other transitions are either g if $|Y| = \emptyset$ or $x = 0$ otherwise.

Lemma 14 *For a given timed automaton A with the set of clocks \mathcal{C} , the timed automaton A' with at most one reset along each transition constructed as above is equivalent to A with respect to the sampling problem.*

Proof. Let $h(\Sigma^* \rightarrow \Sigma^*)$ be a homomorphism with respect to the word concatenation defined by $h(a) = a^{|\mathcal{C}|}, a \in \Sigma$. Clearly, $w \in L(A)$ if and only if $h(w) \in L(A')$. For a run ρ over w in $\llbracket A \rrbracket_{\mathbb{R}_0^+}$, we can construct a run ρ over $h(w)$ in $\llbracket A' \rrbracket_{\mathbb{R}_0^+}$ using the same time delays as ρ by taking no delays in the auxiliary states. For a run ρ over $h(w)$ in $\llbracket A' \rrbracket_{\mathbb{R}_0^+}$, we can construct a run ρ over w in $\llbracket A \rrbracket_{\mathbb{R}_0^+}$ using the delays which are sums of the time delays from ρ by adding up all delays from the auxiliary states. Observe that when at least one clock is reset along a transition in A then the delays in the corresponding auxiliary states are zero. ■

Now we show how to remove the transitions labeled by δ in the extended R-automaton R constructed in Section 5. We use the same algorithm as is used for removing ε -transitions in finite automata. Each sequence of transitions $s_1 \xrightarrow{\delta, (0, \dots, 0)} \dots \xrightarrow{\delta, (0, \dots, 0)} s_{k-1} \xrightarrow{a, (e_1, \dots, e_n)} s_k$ is replaced by the transition $s_1 \xrightarrow{a, (e_1, \dots, e_n)} s_k$. Clearly, this construction results in an extended R-automaton. Let for a word w and an extended R-automaton R , $c_R(w) = \min\{B \mid w \in L_B(R)\}$ (where $\min\{\cdot\} = \omega$). Let $w \upharpoonright \delta$ for $w \in (\Sigma \cup \{\delta\})^*$ denote the projection of w to Σ^* (we skip all letters δ). Let $w \vee w'$ denote shuffle of the two words.

Lemma 15 *Let R be an extended R-automaton constructed in Section 5 and R' be the extended R-automaton constructed as above. Then for each $w \in L(R')$ there is $k \in \mathbb{N}$ and $w' = w \vee \delta^k$ such that $w' \in L(R)$ and $c_{R'}(w) = c_R(w')$. Also, for each $w \in L(R)$, $w \upharpoonright \delta \in L(R')$ and $c_{R'}(w \upharpoonright \delta) \leq c_R(w)$.*

Proof. The proof follows directly from the fact that the effect $(0, \dots, 0)$ does not change the counter values and the preorder \lesssim . ■

The proof of the first item of Theorem 1 follows directly from Lemma 14, Lemma 11, Lemma 12, and Lemma 15.

6.1 ω -sampling

The ω -limitedness problem is decidable for extended R-automata over ω -words with Büchi acceptance conditions. It has been show that ω -universality is decidable for R-automata in [1]. We can use the same trick as for the finite words case to use this result to show that ω -limitedness is decidable for R-automata. Then the decidability of ω -limitedness for extended R-automata follows from Lemmas 6, 7, and 8.

We need to show that the results for finite words can be used to show decidability of sampling for timed automata over ω -words with Büchi acceptance conditions (the second item of Theorem 1).

If the extended R-automaton R constructed from a given timed automaton A is ω -limited then we show that A can be ω -sampled as follows. Let B be such that $L_B^\omega(R) = L^\omega(R)$. Let ρ be an accepting run over $w \in \Sigma^\omega$ with $\max\{\rho\} \leq B$ and let $\varepsilon = 1/(2 \cdot B)$. From Lemma 11 we have that for each (finite) prefix of ρ we have a run ρ' in $\llbracket A \rrbracket_\varepsilon$ along the path in the region graph corresponding to this prefix. Let us denote the set of all such concrete runs H . We show how

for every j we construct a prefix of length j of an infinite concrete run over w in $\llbracket A \rrbracket_\varepsilon$ along the path in the region graph corresponding to ρ . By this we show that there is an infinite accepting run over w in $\llbracket A \rrbracket_\varepsilon$.

First, we define an equivalence relation \sim_K on clock valuations by $v \sim_K v'$ if for all clocks x , $v(x) \neq v'(x)$ implies $v(x) > K$ and $v'(x) > K$. Let K be the greatest constant which appears in A . It is easy to see that for each ε , \sim_K has a finite index on the set of valuations $\{v \mid \forall x \in \mathcal{C} \exists k \in \mathbb{N}. v(x) = k \cdot \varepsilon\}$. Also, $\sim_K \subseteq \cong_K$.

We construct the prefixes inductively. We assume that we can build a prefix of length j ending in a state $\langle q, v \rangle$ such that there is an infinite subset of H of runs whose j -th state is $\langle q, v' \rangle$ for some $v' \sim_K v$. The run of length 0 is just the initial state $\langle q_0, v_0 \rangle$ (which is a prefix of all runs in H). To build the prefix of length $j+1$, we need to extend the prefix of length j . We have infinitely many runs whose j -th state is $\langle q, v' \rangle$ for some $v' \sim_K v$. We pick an infinite subset of these runs such that the valuations in their $j+1$ -st states are equivalent with respect to \sim_K . There is always such an infinite subset, because \sim_K has a finite index in ε -sampled semantics. We pick a state $\langle q', v' \rangle$ such that it can be reached from $\langle q, v \rangle$ and it is equivalent with respect to \sim_K to the states in the infinite subset as the $j+1$ -st state. Clearly, there is such a state.

For the other direction, let us assume that for each B there is $w_B \in \Sigma^\omega$ such that $w_B \notin L_B^\omega(R)$. We show that A cannot be ω -sampled. For each ε we pick $B = 2/\varepsilon$. There is a counter C_{xy} which exceeds B in each accepting run of R over w_B . From Lemma 12, each accepting run of A over w_B requires $\overline{xy}_v \geq (B/2) \cdot \varepsilon = 1$ in some state $\langle q, v \rangle$ along this run. But from the definition, \overline{xy}_v is always strictly smaller than 1.

7 Conclusions

Timed automata with dense time semantics can enforce behaviors, where time distances between events get arbitrarily close to integers while never becoming integral. We have formulated a property distinguishing timed automata which do not use this ability: the untimed language of an automaton in question can be accepted in a semantics where all time delays are multiples of a fixed rational number. These automata preserve all qualitative behaviors (untimed words) when implemented on a platform with a fixed sampling rate. We have also shown that it is decidable whether a timed automaton enjoys this property. The proof characterizes the time differences enforced along runs by a new type of counter automata – Extended R-automata. As a technical contribution of its own interest, we have shown that limitedness is decidable for these automata.

In spite of this positive outcome, our results show a high degree of complexity present in dense time behaviors enforced by strict inequalities. Therefore, when we require from our model that it can be turned into a sampled implementation, we have to consider usage of strict inequalities with a great care.

It is questionable whether the modeling advantages of strict inequalities outweigh the costs of sampling analysis.

Acknowledgements.

We would like to thank Radek Pelánek for valuable comments on the drafts of this paper.

References

- [1] P. A. Abdulla, P. Krcal, and W. Yi. R-automata. In *Proc. of CONCUR'08*, volume 5201 of *LNCS*, pages 67–81. Springer-Verlag, 2008.
- [2] K. Altisen and S. Tripakis. Implementation of timed automata: An issue of semantics or modeling? In *Proc. of FORMATS'05*, volume 3829 of *LNCS*, pages 273–288. Springer-Verlag, 2005.
- [3] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
- [4] E. Asarin, O. Maler, and A. Pnueli. On discretization of delays in timed automata and digital circuits. In *Proc. of CONCUR'98*, volume 1466 of *LNCS*, pages 470–484. Springer-Verlag, 1998.
- [5] M. Bojańczyk and T. Colcombet. Bounds in omega-regularity. In *Proc. of LICS'06*, pages 285–296. IEEE Computer Society Press, 2006.
- [6] T. Colcombet and C. Löding. The nesting-depth of disjunctive μ -calculus for tree languages and the limitedness problem. In *Proc. of CSL'08*, volume 5213 of *LNCS*, pages 416–430. Springer-Verlag, 2008.
- [7] T. Colcombet and C. Löding. The non-deterministic Mostowski hierarchy and distance-parity automata. In *Proc. of ICALP'08*, volume 5126 of *LNCS*, pages 398–409. Springer-Verlag, 2008.
- [8] T. A. Henzinger, F. Cassez, and J.-F. Raskin. A comparison of control problems for timed and hybrid systems. In *Proc. of HSCC'02*, volume 2289 of *LNCS*, pages 134–148. Springer-Verlag, 2002.
- [9] A. Göllü, A. Puri, and P. Varaiya. Discretization of timed automata. In *Proc. of CDC'94*, pages 957–958, 1994.
- [10] Kosaburo Hashiguchi. Limitedness theorem on finite automata with distance functions. *Computer and System Sciences*, 24(2):233–244, 1982.
- [11] Kosaburo Hashiguchi. Improved limitedness theorems on finite automata with distance functions. *Theoretical Computer Science*, 72(1):27–38, 1990.
- [12] D. Kirsten. Distance desert automata and the star height problem. *Informatique Theorique et Applications*, 39(3):455–509, 2005.

- [13] P. Krčál, L. Mokrushin, P. S. Thiagarajan, and W. Yi. Timed vs. time triggered automata. In *Proc. of CONCUR'04*, volume 3170 of *LNCS*, pages 340–354. Springer-Verlag, 2004.
- [14] P. Krčál and R. Pelánek. On sampled semantics of timed systems. In *Proc. of FSTTCS'05*, volume 3821 of *LNCS*, pages 310–321. Springer-Verlag, 2005.
- [15] Hing Leung. Limitedness theorem on finite automata with distance functions: an algebraic proof. *Theoretical Computer Science*, 81(1):137–145, 1991.
- [16] J. Ouaknine and J. Worrell. Universality and language inclusion for open and closed timed automata. In *Proc. of HSCC'03*, volume 2623 of *LNCS*, pages 375–388. Springer-Verlag, 2003.
- [17] I. Simon. On semigroups of matrices over the tropical semiring. *Informatique Theorique et Applications*, 28(3-4):277–294, 1994.
- [18] Z. Manna T. A. Henzinger and A. Pnueli. What good are digital clocks? In *Proc. of ICALP'92*, volume 623 of *LNCS*, pages 545–558. Springer-Verlag, 1992.
- [19] M. De Wulf, L. Doyen, and J.-F. Raskin. Almost ASAP semantics: From timed models to timed implementations. In *Proc. of HSCC'04*, volume 2993 of *LNCS*, pages 296–310. Springer-Verlag, 2004.

Paper IV



Communicating Timed Automata: The More Synchronous, the More Difficult to Verify*

Pavel Krcal and Wang Yi

Department of Information Technology
Uppsala University, Sweden
Email: {pavelk, yi}@it.uu.se

Abstract. We study channel systems whose behaviour (sending and receiving messages via unbounded FIFO channels) must follow given timing constraints specifying the execution speeds of the local components. We propose Communicating Timed Automata (CTA) to model such systems. The goal is to study the borderline between decidable and undecidable classes of channel systems in the timed setting. Our technical results include: (1) CTA with one channel without shared states in the form $(A_1, A_2, c_{1,2})$ is equivalent to one-counter machine, implying that verification problems such as checking state reachability and channel boundedness are decidable, and (2) CTA with two channels without sharing states in the form $(A_1, A_2, A_3, c_{1,2}, c_{2,3})$ has the power of Turing machines. Note that in the untimed setting, these systems are no more expressive than finite state machines. We show that the capability of synchronizing on time makes it substantially more difficult to verify channel systems.

1 Introduction

FIFO channels (i.e., unbounded buffers) are widely used as a communication mechanism in concurrent systems. In many applications, channels are a critical element for the correct functioning of such systems. In this work, we study timed systems whose components communicate through (unbounded) channels. An example of such systems is illustrated in Figure IV.1, where A_1 is a producer (or sender) which generates messages and puts them into the buffer $c_{1,2}$ and A_2 is a consumer (or receiver) which gets messages from the buffer. Assume that the production and consumption of messages must follow given timing constraints (specifying the relative execution speeds of the producer and the consumer). A relevant question to ask is whether the channel is bounded, and if it is, what is the maximal size of the buffer. This is a typical scenario in designing embedded systems, where it is desirable to know a

*This work is partially supported by the European Research Training Network GAMES.

priori the maximal size of a buffer needed to avoid buffer overflow and over-allocation of memory blocks in the final implementation.

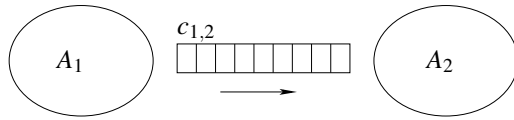


Figure IV.1: A schema of a CTA with one channel.

In the literature, channel systems have been studied intensively in the untimed setting, within the context of verification of infinite state systems (see below for related work which provides a brief summary of known results). To our best knowledge, this is the first attempt to study channel systems in the timed setting. The existing works address mainly channel systems that are a finite set of Communicating Finite State Machines (CFSMs). In the CFSM model, no notion of time is assumed and systems run in a fully asynchronous manner in the sense that any local move of a machine is allowed at any time. We observe that for systems modeled as CFSMs, the source of infiniteness is in not only *unbounded channels* but also the capability of *synchronization* or exchanging information between the machines. In fact, asynchronous systems – as illustrated in Figure IV.1 and IV.2 with only one-directional communication, where the receivers are not allowed to inform directly or indirectly the senders about the receipt of messages – are no more expressive than finite state machines [Pac03, CF], and thus all properties such as reachability and channel boundedness are decidable. Roughly speaking, synchronization within CFSMs may be achieved through either shared states [BZ83], or two-direction communication [FM97] or combination of accepting conditions and doubled one-direction channels [Pac03, Pac82]. The synchronization features together with the unboundedness of channels are the essential source of undecidability for channel systems in the untimed setting.

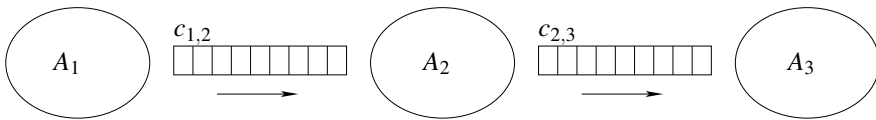


Figure IV.2: A schema of a CTA with two channels.

We shall see that in the timed setting, the implicit synchronization of system components on the global time as well as the density of time will add yet another dimension of infiniteness for channel systems. In this paper, we shall study Communicating Timed Automata (CTA), i.e., networks of timed automata extended with (unbounded) channels. A CTA is a channel system where the sending and receiving transitions of machines are constrained with clock constraints. We shall show that channel systems (with one channel) as

illustrated in Figure IV.1, which accept only regular languages in the untimed setting, are expressive enough to simulate one-counter machines in the timed setting. However, the density of time adds no more expressive power (than discrete time), and many questions of interests such as reachability and channel boundedness are still decidable for CTA with one channel. As a main technical contribution, we present a novel proof showing that CTAs with one channel without sharing states are no more expressive than one-counter machines. The proof uses the notion of CDR (Clock Difference Relations) developed in [KP05b]. To study the borderline of decidability and undecidability for CTA, we have shown that CTAs with two channels, as illustrated in Figure IV.2, can simulate Turing Machines.

Related Work

Channel systems, i.e., networks of communicating finite state machines (*CF-SMs*) have been widely studied in the untimed setting, as a model for communication protocols, in which no global notion of time is assumed and any local move of FSMs at any time is allowed. The first undecidability results for the untimed setting were presented in [BZ83] showing that two FSMs with shared states and one channel can simulate Turing machines. Further results consider even more restricted settings, showing that two identical simple FSMs with one channel in both directions are powerful enough to simulate a Turing machine [FM97]. A surprising result due to [Pac03, Pac82] is that two FSMs connected by two channels going in the same direction can simulate Initial Post's Correspondence Problem, and therefore have the power of Turing machine. Classes of CFMSs with decidable reachability problems have been identified in [CF] (half-duplex systems), [Pac03] (cyclic systems with one channel bounded), and [PP92] (cyclic systems with one-type messages). Abstractions of CFMSs for acceleration in reachability analysis are presented in [FPS03]. Another recent work [GMK04] shows the equivalence of several formalisms when the communication is existentially bounded. Apart from work on systems with perfect channels, systems with lossy channels have been studied in [AJ96a, AJ96b]. An excellent survey on work in this direction can be found in [CFP96].

2 Communicating Timed Automata

In this section we present the syntax and semantics of Communicating Timed Automata (CTA). A CTA is essentially a channel system modeled as a CFMS where finite state machines are replaced by timed automata. In the following, we assume that the reader is familiar with notions related to timed automata [AD94].

Syntax.

A network of Communicating Timed Automata (CTA) is a tuple $(A_1, A_2, \dots, A_n, c_{i_1, j_1}, c_{i_2, j_2}, \dots, c_{i_m, j_m})$ where each $A_i = (Q_i, \Sigma, \mathcal{C}_i, E_i, q_i^0, F_i)$ is a timed

automaton and each $c_{i,j}, i, j \in \{1 \dots n\}$ is an unidirectional unbounded channel containing messages sent from A_i to A_j . Mutually disjoint finite sets Q_1, \dots, Q_n contain locations of A_i 's. A finite set Σ denotes a communication alphabet common for all A_i 's. In addition, we assume that automata may perform an internal transition denoted by ε . \mathcal{C}_i is a finite set of real-valued clocks ($\mathcal{C}_i, \mathcal{C}_j$ are disjoint for $i \neq j$), $q_i^0 \in Q_i$ is an initial location, and $F_i \subseteq Q_i$ is a set of accepting locations. $E_i \subseteq Q_i \times (\{1 \dots n\} \times \{?, !\} \times \Sigma) \cup \{\varepsilon\} \times \mathcal{G}(\mathcal{C}) \times 2^{\mathcal{C}} \times Q_i$ is the set of transitions of A_i . Transitions are labeled by not only a letter from Σ , but also information about whether a letter is sent or received (! or ?, respectively) and to or from which channel. E.g., a transition $(q_1, (4!a), \emptyset, \emptyset, q_2)$ of A_3 means that A_3 can move from the location q_1 to the location q_2 sending a into the channel $c_{3,4}$ (the transition is not guarded and does not reset any clocks). When such a transition is taken a letter a is put into the channel $c_{3,4}$. A transition $(q_3, (2?b), \emptyset, \emptyset, q_4)$ of A_1 means that A_1 can move from q_3 to q_4 and read b from $c_{2,1}$. Such a move is possible only when there is a letter b at the head of $c_{2,1}$. We write $q_i \xrightarrow{k!a,g,r} q'_i$ when $(q_i, k!a, g, r, q'_i) \in E$. Channels are assumed to be perfect. We will denote the contents of a channel by finite words over Σ .

Note that there can be pairs of timed automata which are not connected by a channel (e.g., if there is no $c_{1,2}$ for automata A_1, A_2). We assume that there can also be channels from a timed automaton to itself. E.g., a system $(A_1, c_{1,1})$ can serve as a model of two timed automata with *shared states* connected by a channel.

Semantics.

Let $v_i : \mathcal{C}_i \mapsto \mathbb{R}_{\geq 0}$ denote a valuation of clocks in A_i . Let $v_i \models g$ denote that the guard g is satisfied by v_i and $r(v_i), r \subseteq \mathcal{C}_i$ denote a valuation where all clocks from r are reset and other clocks keep their values. A state of the system is a tuple $(q_1, v_1, \dots, q_n, v_n, w_1, \dots, w_m)$, where $q_i \in Q_i$ is a location of A_i and $w_k \in \Sigma^*$ is the content of channel c_{i_k, j_k} . We define the semantics of CTA based on LTS.

Definition 1 (Synchronized Semantics) *The semantics of a CTA $(A_1, \dots, A_n, c_{i_1, j_1}, \dots, c_{i_m, j_m})$ is a labeled transition system with initial state $(q_1^0, v_1^0, q_2^0, v_2^0, \dots, q_n^0, v_n^0, \varepsilon, \dots, \varepsilon)$, where $v_i^0(x) = 0$ for all $x \in \mathcal{C}_i$ and two types of transitions – time pass and discrete transition – defined as follows. Let $s = (q_1, v_1, \dots, q_n, v_n, w_1, \dots, w_m)$ and $s' = (q'_1, v'_1, \dots, q'_n, v'_n, w'_1, \dots, w'_m)$.*

- $s \xrightarrow{t} s'$ if $v'_i = v_i + t$, $q'_i = q_i$ and $w'_j = w_j$ for all $1 \leq i \leq n$ and $1 \leq j \leq m$.
- $s \xrightarrow{(a,i,k,!)} s'$ if $q_i \xrightarrow{k!a,g,r} q'_i$, $w'_l = a \cdot w_l$, where w_l is the content of $c_{i,k}$, $v_i \models g$, $v'_i = r(v)$, and $q'_j = q_j, v'_j = v_j, w'_k = w_k$ for all $j \neq i, k \neq l$,
- $s \xrightarrow{(a,i,k,?)} s'$ if $q_i \xrightarrow{k?a,g,r} q'_i$, $a \neq \varepsilon$, $w'_l \cdot a = w_l$, where w_l is the content of $c_{k,i}$, $v_i \models g$, $v'_i = r(v)$, and $q'_j = q_j, v'_j = v_j, w'_k = w_k$ for all $j \neq i, k \neq l$, and

- $s \xrightarrow{(\varepsilon, i)} s'$ if $q_i \xrightarrow{\varepsilon, g, r} q'_i$, $v_i \models g$, $v' = r(v)$, $q'_j = q_j$, $v'_j = v_j$, $w'_k = w_k$ for all $j \neq i, k \in \{1, \dots, m\}$, and there is no $q_i \xrightarrow{k?a, \bar{g}, \bar{r}} q''_i$ such that $v_i \models \bar{g}$ and $w_l = w''_l \cdot a$, where w_l is the content of $c_{k,i}$.

All automata move synchronously; time passes at the same pace for all of them. The automata read from the channels in an *urgent* manner, an automaton is not allowed to take an ε -transition if it can take a receiving a -transition and a is at the head of the corresponding channel. Another possibility is to define reading as non-urgent, i.e., there are no restrictions on taking ε transitions. In Section 3, we show by an example that CTA's even with non-urgent reading from the channels will have strictly more expressive power than CFSMs in the untimed setting.

Let S be a CTA and T_S be its corresponding LTS. By ρ we denote a finite path in T_S , by $[\rho]$ a sequence of labels occurring along ρ , and by $[\rho]_i^!$ ($[\rho]_i^?$) a sequence of letters from Σ which is a projection of $[\rho]$ to letters sent (received) by an automaton A_i . If the location vector (q_1, \dots, q_n) of the last global state of ρ is accepting (i.e., $\forall i. q_i \in F_i$) then we say that the run is accepting, denoted $\rho \triangleright T_S$. A language accepted by a CTA S is a set $L_S(S) = \{[\rho]_1^! \mid \rho \triangleright T_S\}$.

The two groups of problems have usually been studied for CFSMs. The first group contains *reachability* problems – state reachability, control vector reachability, location vector reachability, deadlock (a state where all automata can only read and the channels are empty), unspecified reception (an automaton can only read, but no channel contains a matching letter), and stability (all channels are empty). The second type of problems is *boundedness* problems – whether the set of all reachable contents of all channels is finite or whether the set all reachable contents of a given channel is finite (strong boundedness).

Note that we can model CFSMs by CTA. Therefore, all negative results proved for CFSMs apply also to our model. In the following, we study the expressive power of the model by identifying decidable and undecidable classes of CTA.

3 CTA with One Channel

Let us first consider a system $(A_1, A_2, c_{1,2})$ schematically depicted in Figure IV.1. It has been shown that CFSMs with such topology accept regular languages and reachability and boundedness problems are decidable [Pac03, CF]. We show that CTA of this form can accept also some non-regular context-free languages. Moreover, we show that for such a CTA there is a one-counter machine which accepts the same language. Therefore, state reachability and channel boundedness problems are decidable, which follows from the decidability of emptiness and infiniteness for context-free languages.

To establish the proof, we propose an alternative (*desynchronized concrete*) semantics for CTA which resembles the reordering technique [Pac03] for CFSMs and the local time semantics for timed systems [BJLY98]. However,

states in this semantics still contain concrete valuations of clocks. Therefore, we define a (*desynchronized symbolic*) semantics where the continuous part of the state has a finite symbolic representation. This symbolic semantics can be easily simulated by a one-counter machine. We also show that instructions of a one-counter machine can be simulated by a CTA of the form $(A_1, A_2, c_{1,2})$ and thus the expressive power of CTA with this topology is equivalent to one-counter machine.

Intuitively, we let the automata to desynchronize so that there is at most one message in the channel during the first part of the computation and that only the producing automaton runs during the second part of the computation. Local time (time from the beginning of the computation) can be different in A_1 and A_2 . We keep track of the difference between local times of automata in a real valued variable. The acceptance condition is extended by a requirement that the system is be synchronized, i.e., the value of this variable should be equal to 0.

In the following, we denote A_1 as A and A_2 as B . We also write $!a$ instead of $2!a$ and $?a$ instead of $1?a$. Without loss of generality, we assume that there is a clock t_i in each A_i which is never reset. The reason is to simplify the notation later. A state in the concrete desynchronized semantics is a tuple $(q_A, v_A, q_B, v_B, w, T)$, where $q_A \in Q_A, q_B \in Q_B, w \in \Sigma^*$, valuations v_A, v_B are as in the original semantics, and $T \in \mathbb{R}$ is the lag of B behind A (it is negative if B is ahead). We need to take special care about reading – a letter should not be read before it has been produced.

We let the automata to alternate in running as long as the size of the channel content does not exceed 1. When it contains at least two letters then only A can move. We assume $a \in \Sigma$ and $w \in \Sigma^*$ in the following definition.

Definition 2 (Desynchronized Concrete Semantics) *The desynchronized concrete semantics of a CTA $(A, B, q_{A,B})$ is a labeled transition system with initial state $(q_A^0, v_A^0, q_B^0, v_B^0, \varepsilon, 0)$ and transitions induced by the following rules:*

- $(q_A, v_A, q_B, v_B, w, T) \xrightarrow{t}_{dc} (q_A, v'_A, q_B, v_B, w, T + t)$ if $(q_A, v_A) \xrightarrow{t} (q_A, v_A + t)$,
- $(q_A, v_A, q_B, v_B, w, T) \xrightarrow{(a,1,2,!)}_{dc} (q'_A, v'_A, q_B, v_B, a \cdot w, T)$ if $(q_A, v_A) \xrightarrow{!a} (q'_A, v'_A)$,
- $(q_A, v_A, q_B, v_B, w, T) \xrightarrow{t}_{dc} (q_A, v_A, q_B, v'_B, w, T - t)$ if $(q_B, v_B) \xrightarrow{t} (q_B, v_B)$ and $|w| \leq 1$,
- $(q_A, v_A, q_B, v_B, a, T) \xrightarrow{(a,2,1,?) }_{dc} (q_A, v_A, q'_B, v'_B, \varepsilon, T)$ if $(q_B, v_B) \xrightarrow{?a} (q'_B, v'_B)$ and $T \leq 0$,
- $(q_A, v_A, q_B, v_B, w, T) \xrightarrow{\varepsilon}_{dc} (q_A, v_A, q'_B, v'_B, w, T)$ if $(q_B, v_B) \xrightarrow{\varepsilon} (q'_B, v'_B)$, $|w| \leq 1$, if $T \geq 0$ then $(w = a \Rightarrow (q_B, v_B) \xrightarrow{?a})$, and if $T < 0$ then $(w = a \wedge (q_B, v_B) \xrightarrow{?a})$.

A run with the last state $(q_A, v_A, q_B, v_B, w, T)$ is accepting if $q_A \in F_A, q_B \in F_B$, and $T = 0$. Definition of the accepted language $L_{DC}(S)$ for a given CTA S is the same as for synchronized semantics. The set of reachable states of a

given CTA is equal to the set of states reachable in its desynchronized concrete semantics where $T = 0$. Also, the language accepted by a CTA is the same in both semantics.

Lemma 1 *For a given CTA S of the form $(A, B, c_{A,B})$, the reachability set $\{(q_A, v_A, q_B, v_B, w) \mid (q_A^0, v_A^0, q_B^0, v_B^0, \varepsilon) \rightarrow^* (q_A, v_A, q_B, v_B, w)\}$ is equal to the set $\{(q_A, v_A, q_B, v_B, w) \mid (q_A^0, v_A^0, q_B^0, v_B^0, \varepsilon, 0) \xrightarrow{dc^*} (q_A, v_A, q_B, v_B, w, 0)\}$. Moreover, $L_S(S) = L_{DC}(S)$.*

The basic idea of the proof of this lemma is the same as in [Pac03]. Desynchronized concrete semantics cannot reach more states where $T = 0$ or accept more words because the counter gives us a possibility to check the following conditions on the transitions of B . A letter can be read only after it has been produced and ε -transitions can be taken only when no enabled transition is labeled by the head of the buffer.

The desynchronization semantics shows how to avoid necessity to remember the whole content of the buffer during the run of a CTA. Note that one does not have to remember the content of the channel when its size exceeds 1, because it will never be read. The price we have to pay is an additional real number as a part of the state. In case of discrete time, T is an integer and therefore one can replace such a system by a bisimilar one-counter machine. To be able to do the same for the dense time, we need to handle the real time and T in a symbolic way, such that we get a finite state control unit and one counter.

We use standard regions [AD94] with a small modification. The order of the fractional parts is remembered even for the clocks whose value is above the greatest constant. We remember some distinguished value ($> K$) instead of the value of the integral part. Regions will be denoted by D, D_A, D_B . Let $integral(D)$ denote clocks that have zero fractional part in D , $fr(x)$ denote the fractional part of a real number x . We say that $D(x) > D(y)$ if for all valuations $v \in D$ it holds that $v(x) > v(y)$. When D is a region over clocks of two automata A and B then by $(v_A, v_B) \in D$ we mean that $v \in D$ where $v(x) = v_A(x)$ for all $x \in \mathcal{C}_A$ and $v(y) = v_B(y)$ for all $y \in \mathcal{C}_B$. We write $D \Rightarrow D_A$ if D is a region over clocks of A, B , D_A is a region over clocks of A , and for all $(v, v') \in D$ it holds that $v \in D_A$.

We also need to take care of T . There are two sources of infinity in T – its integral part, which can grow arbitrarily large, and its fractional part. We remember the integral part of T in a counter, denoted N . To remember the fractional part of T , we use the extra local clocks t_A and t_B of A and B . We observe that the difference of their fractional parts is equal to the fractional part of T (we do not use their integral parts). More precisely, if $(q_A, v_A, q_B, v_B, w, T)$ is reachable and $N = \lceil T \rceil$ then $T = N + (fr(v_A(t_A)) - fr(v_B(t_B)))$ if $v_A(t_A) \geq v_B(t_B)$ and $T = N + (1 - (fr(v_B(t_B)) - fr(v_A(t_A))))$ if $v_A(t_A) < v_B(t_B)$. We write that $T = \text{Sum}(N, t_A, t_B)$.

The fractional parts of t_A and t_B are then symbolically represented by regions and we remember their relative order as a constraint of the form $t_A \bowtie t_B$,

where $\bowtie \in \{<, =, >\}$. Assume that local regions D_A, D_B were reached during the standard reachability analysis. For two given local regions D_A, D_B , our goal is to find a global region D which contains only valuations reachable in the desynchronized concrete semantics. We can define D as an ordering of the fractional parts of clocks which is consistent with D_A, D_B ($D \Rightarrow D_A, D \Rightarrow D_B$), and with $t_A \bowtie t_B$.

However, such symbolic representation is not sufficient. There are CTA for which symbolic analysis reaches $D_A, D_B, t_A \bowtie t_B$, but there is a global region D consistent with $D_A, D_B, t_A \bowtie t_B$ which contains unreachable valuations. As a counterexample, consider the system in Figure IV.3. The ε -transition together with the urgency makes sure that x is reset earlier than y . Then the guard $x = 1$ is satisfied strictly earlier than the guard $y = 1$. But A produces the second a exactly when $y = 1$. B resets x when it is equal to 1 and it can reset v only after the second a has been produced. Therefore, there is a non-zero delay between reading of the two a 's in B . This means that the guards $x = 1$ and $v = 1$ cannot be satisfied at the same time. However, this is possible in our naive symbolic semantics, because the fact that x is reset strictly earlier than y is lost (they belong to different automata). Thus, the other a can arrive also when $x = 1$.

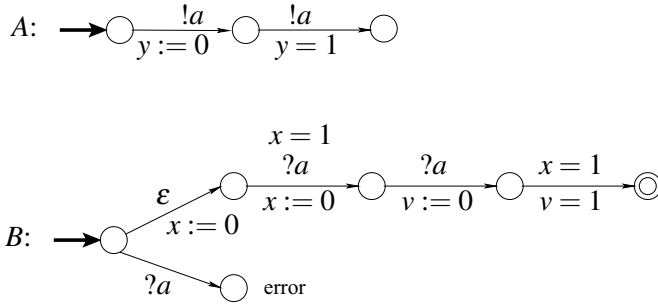


Figure IV.3: A CTA illustrating the need of additional constraints on the symbolic state. The accepting state in B is not reachable, but it can be reached in a naive symbolic semantics.

To cope with this problem, we add more information to the symbolic state. The information that the value of x is strictly greater than y when y is reset is not very useful, because due to the desynchronization the order of the clock values can change in time. A suitable notion is the difference between the clock values, which does not change in time. Now we can benefit from the fact that t_A and t_B are never reset and relate all other clocks to them. We use the concept of *clock difference relations*, which has been introduced in [KP05b] to characterize reachability relations. Here we give a slightly modified definition which suits our purposes better. To differentiate this definition from the original one, we call it *desynchronized clock difference relations* here, but later we will use only an abbreviation CDR or clock difference relation.

Definition 3 A desynchronized clock difference relation (CDR) is a set of (in)equalities of the form

- $exp \bowtie exp$
- $exp \bowtie 1 - (exp)$

where exp is a clock difference (over the clocks of either A or B) in the form: $t_A - x$, $x - t_A$, $t_B - y$ or $y - t_B$, x is a clock of A , y is a clock of B , and $\bowtie \in \{<, >, =\}$.

Definition 4 *The semantics of a CDR is defined as follows. Assume C is a CDR. We say that a pair of valuations (v, v') satisfies C ($(v, v') \models C$) if and only if:*

- if $x - y \bowtie u - v \in C$ then $fr(v(x)) - fr(v(y)) \bowtie fr(v'(u)) - fr(v'(v))$,
- if $x - y \bowtie 1 - (u - v) \in C$ then $fr(v(x)) - fr(v(y)) \bowtie 1 - (fr(v'(u)) - fr(v'(v)))$,

Additionally, we require that for each $x - y$ (or $u - v$), $fr(v(x)) - fr(v(y)) > 0$.

We will use clock difference relations to restrict possible merges of regions over clocks of A and B . The merged regions represent only reachable concrete desynchronized valuations now.

States of the desynchronized symbolic system $(q_A, D_A, q_B, D_B, C, t_A \bowtie t_B, w, N)$ consist of locations and regions of A and B , respectively, clock difference relations, relation of t_A and t_B , $w \in \Sigma^*$ is a content of the buffer, and N is an integer used to remember the difference between the integral parts of t_A and t_B .

Also, we assume without loss of generality that there is a special clock x_A^{zero}, x_B^{zero} in A, B , respectively, which is reset always when the other automaton starts to move (at the end of automaton's moves). This means that each sequence of moves of one automaton is finished by the reset transition for clock x_A^{zero} or x_B^{zero} . These clocks are used to keep the information about the distance of the fractional part of t_A, t_B from 0 and they are necessary for the correctness with CDRs of such a simple form.

We need one more technical definition before the definition of the semantics. We define a predicate $Consistent(D_A, D_B, C, t_A \bowtie t_B) = \exists (v_A, v_B) \cdot v_A(t_A) \bowtie v_B(t_B) \wedge (v_A, v_B) \models C \wedge v_A \in D_A \wedge v_B \in D_B$.

Definition 5 (Desynchronized Symbolic Semantics) *The desynchronized symbolic semantics of a CTA $(A, B, q_{A,B})$ is a labeled transition system with initial state $(q_A^0, D_A^0, q_B^0, D_B^0, \emptyset, t_A = t_B, \varepsilon, 0)$ and transitions described in Table IV.1, Table IV.2, Table IV.3, and Table IV.4.*

A run with the last state $(q_A, D_A, q_B, D_B, C, t_A \bowtie t_B, w, N)$ is accepting if $q_A \in F_A, q_B \in F_B, N = 0$, and $t_A = t_B$. Definition of the accepted language $L_{DS}(S)$ for a given CTA S is the same as for synchronized semantics.

Correctness.

Now we state that the desynchronized symbolic semantics is reachability and language equivalent to the desynchronized concrete one. At first, we define

Table IV.1: Rules for symbolic transitions induced by the region graph of A . For clarity, we omit locations in the rules for time pass.

Time Pass:	
$D_A \rightarrow D'_A, \exists x \in \text{integral}(D_A), C' = C$	
$(D_A, D_B, C, t_A < t_B, w, N)$	$\xrightarrow{ds} (D'_A, D_B, C', t_A < t_B, w, N)$
$(D_A, D_B, C, t_A = t_B, w, N)$	$\xrightarrow{ds} (D'_A, D_B, C', t_A > t_B, w, N)$
$(D_A, D_B, C, t_A > t_B, w, N)$	$\xrightarrow{ds} (D'_A, D_B, C', t_A > t_B, w, N)$
$D'_A = D_A, \nexists x \in \text{integral}(D_A), C' = C$	
$(D_A, D_B, C, t_A < t_B, w, N)$	$\xrightarrow{ds} (D'_A, D_B, C', t_A = t_B, w, N + 1)$ if Consistent($D'_A, D_B, C', t_A = t_B$)
$(D_A, D_B, C, t_A = t_B, w, N)$	$\xrightarrow{ds} (D'_A, D_B, C', t_A > t_B, w, N)$
$D_A \rightarrow D'_A, \exists x \in \text{integral}(D'_A), C'$ is computed according to Table IV.3	
$(D_A, D_B, C, t_A < t_B, w, N)$	$\xrightarrow{ds} (D'_A, D_B, C', t_A = t_B, w, N + 1)$ if Consistent($D'_A, D_B, C', t_A = t_B$)
$(D_A, D_B, C, t_A < t_B, w, N)$	$\xrightarrow{ds} (D'_A, D_B, C', t_A < t_B, w, N)$ if Consistent($D'_A, D_B, C', t_A < t_B$)
$(D_A, D_B, C, t_A > t_B, w, N)$	$\xrightarrow{ds} (D'_A, D_B, C', t_A > t_B, w, N)$ if $t_A \notin \text{integral}(D'_A)$
$(D_A, D_B, C, t_A > t_B, w, N)$	$\xrightarrow{ds} (D'_A, D_B, C', t_A < t_B, w, N)$ if $t_A \in \text{integral}(D'_A), t_B \notin \text{integral}(D_B)$
$(D_A, D_B, C, t_A > t_B, w, N)$	$\xrightarrow{ds} (D'_A, D_B, C', t_A = t_B, w, N + 1)$ if $t_A \in \text{integral}(D'_A), t_B \in \text{integral}(D_B)$
Discrete Transition:	
$(q_A, D_A) \rightarrow (q'_A, D'_A), x$ is reset, C' is computed according to Table IV.3	
$(q_A, D_A, q_B, D_B, C, t_A \bowtie t_B, w, N)$	$\xrightarrow{ds}^{(a,1,2,!)} (q'_A, D'_A, q_B, D_B, C', t_A \bowtie t_B, a \cdot w, N)$ if $a \in \Sigma \cup \{\varepsilon\}$ is the label on the corresponding edge of A
$(q_A, D_A) \rightarrow (q'_A, D_A),$ no clock is reset, $C' = C$	
$(q_A, D_A, q_B, D_B, C, t_A \bowtie t_B, w, N)$	$\xrightarrow{ds}^{(a,1,2,!)} (q'_A, D_A, q_B, D_B, C', t_A \bowtie t_B, a \cdot w, N)$ if $a \in \Sigma \cup \{\varepsilon\}$ is the label on the corresponding edge of A

Table IV.2: Rules for symbolic transitions induced by the region graph of B . All transitions are constrained by $|w| \leq 1$. For clarity, we omit locations in the rules for time pass.

Time Pass:	
$D_B \rightarrow D'_B, \exists x \in \text{integral}(D_B), C' = C$	
$(D_A, D_B, C, t_A < t_B, w, N) \longrightarrow_{ds}$	$(D_A, D'_B, C', t_A < t_B, w, N)$
$(D_A, D_B, C, t_A = t_B, w, N) \longrightarrow_{ds}$	$(D_A, D'_B, C', t_A < t_B, w, N - 1)$
$(D_A, D_B, C, t_A > t_B, w, N) \longrightarrow_{ds}$	$(D_A, D'_B, C', t_A > t_B, w, N)$
$D'_B = D_B, \nexists x \in \text{integral}(D_B), C' = C$	
$(D_A, D_B, C, t_A > t_B, w, N) \longrightarrow_{ds}$	$(D_A, D'_B, C', t_A = t_B, w, N)$ if Consistent($D_A, D_B, C, t_A = t_B$)
$(D_A, D_B, C, t_A = t_B, w, N) \longrightarrow_{ds}$	$(D_A, D'_B, C', t_A < t_B, w, N - 1)$
$D_B \rightarrow D'_B, \exists x \in \text{integral}(D'_B), C'$ is computed according to Table IV.4	
$(D_A, D_B, C, t_A > t_B, w, N) \longrightarrow_{ds}$	$(D_A, D'_B, C', t_A = t_B, w, N)$ if Consistent($D_A, D'_B, C', t_A = t_B$)
$(D_A, D_B, C, t_A > t_B, w, N) \longrightarrow_{ds}$	$(D_A, D'_B, C', t_A > t_B, w, N)$ if Consistent($D_A, D'_B, C', t_A > t_B$)
$(D_A, D_B, C, t_A < t_B, w, N) \longrightarrow_{ds}$	$(D_A, D'_B, C', t_A < t_B, w, N)$ if $t_B \notin \text{integral}(D'_B)$
$(D_A, D_B, C, t_A < t_B, w, N) \longrightarrow_{ds}$	$(D_A, D'_B, C', t_A > t_B, w, N)$ if $t_B \in \text{integral}(D'_B), t_A \notin \text{integral}(D_A)$
$(D_A, D_B, C, t_A < t_B, w, N) \longrightarrow_{ds}$	$(D_A, D'_B, C', t_A = t_B, w, N)$ if $t_B \in \text{integral}(D'_B), t_A \in \text{integral}(D_A)$
Discrete Transition:	
$(q_B, D_B) \rightarrow (q'_B, D'_B), x$ is reset, C' is computed according to Table IV.4	
$(q_A, D_A, q_B, D_B, C, t_A \bowtie t_B, a, N) \xrightarrow{(a, 2, 1, ?)}_{ds}$	$(q_A, D_A, q'_B, D'_B, C', t_A \bowtie t_B, \varepsilon, N)$ if $?a, a \in \Sigma$, is the label on the corr. edge of B , and $N < 0 \vee (N = 0 \wedge t_A = t_B)$
$(q_A, D_A, q_B, D_B, C, t_A \bowtie t_B, w, N) \xrightarrow{\varepsilon}_{ds}$	$(q_A, D_A, q'_B, D'_B, C', t_A \bowtie t_B, w, N)$ if ε is the label on the corr. edge of B , if $N \geq 0$ then $(w = a \Rightarrow (q_B, v_B) \xrightarrow{?a})$ and if $N < 0$ then $(w = a \wedge (q_B, v_B) \xrightarrow{?a})$

Similarly when no clock is reset.

Table IV.3: Updates of the clock difference relations according to the type of the transition of the automaton A of the desynchronized symbolic system. We write e for a clock difference relation (a single (in)equality). We write exp for an expression of the form $x - y$ or $1 - (x - y)$ where x, y are clock from the automaton given by the context.

C'	Condition, A moves
$D_A \rightarrow D'_A, \exists x \in \text{integral}(D'_A)$	
e $y - x \bowtie^{-1} 1 - (exp)$	$e \in C, e$ does not contain any $x \in \text{integral}(D'_A)$ $x - y \bowtie exp \in C, x \in \text{integral}(D'_A)$
$D_A \rightarrow D'_A, x$ is reset	
e $t_A - x > exp$ $t_A - x < 1 - exp$ $t_A - x < t_B - y$ $t_A - x < 1 - (y - t_B)$ $t_A - x = t_B - y$ $t_A - x > t_B - y$ $t_A - x < 1 - (y - t_B)$ $t_A - x > t_B - y$	$e \in C, e$ does not contain x $t_A - y \geq exp \in C$ $z - t_A \geq exp \in C$ $t_A < t_B, y \in \text{integral}(D_B)$ $t_A < t_B, D_B(y) > D_B(t_B)$ $t_A = t_B, y \in \text{integral}(D_B)$ $t_A = t_B, y \notin \text{integral}(D_B), D_B(y) < D_B(t_B)$ $t_A = t_B, D_B(y) > D_B(t_B)$ $t_A > t_B, D_B(y) < D_B(t_B)$

Table IV.4: Updates of the clock difference relations according to the type of the transition of the automaton B of the desynchronized symbolic system. We write e for a clock difference relation (a single (in)equality). We write exp for an expression of the form $x - y$ or $1 - (x - y)$ where x, y are clock from the automaton given by the context.

C'	Condition, B moves
$D_B \rightarrow D'_B, \exists x \in \text{integral}(D'_B)$	
e $exp \bowtie y - x$ $exp \bowtie 1 - (y - x)$	$e \in C, e$ does not contain any $x \in \text{integral}(D'_B)$ $exp \bowtie 1 - (x - y) \in C, x \in \text{integral}(D'_B)$ $exp \bowtie x - y \in C, x \in \text{integral}(D'_B)$
$D_B \rightarrow D'_B, x$ is reset	
e $exp < t_B - x$ $exp < 1 - (t_B - x)$ $t_A - y > t_B - x$ $y - t_A < 1 - (t_B - x)$ $t_A - y = t_B - x$ $t_A - y < t_B - x$ $y - t_A < 1 - (t_B - x)$ $t_A - y < t_B - x$	$e \in C, e$ does not contain x $exp \leq t_B - y \in C$ $exp \leq z - t_B \in C$ $t_A > t_B, y \in \text{integral}(D_A)$ $t_A > t_B, D_A(y) > D_A(t_A)$ $t_A = t_B, y \in \text{integral}(D_A)$ $t_A = t_B, y \notin \text{integral}(D_A), D_A(y) < D_A(t_A)$ $t_A = t_B, D_A(y) > D_A(t_A)$ $t_A < t_B, D_A(y) < D_A(t_A)$

an equivalence on valuations which takes care about the clocks whose value is above the greatest constant. By $v \sim_K v'$ we mean that $\text{fr}(v(x)) = \text{fr}(v'(x))$ and moreover $v(x) = v'(x)$ for all clocks x such that $v(x) \leq K$.

Lemma 2 *If $(q_A^0, v_A^0, q_B^0, v_B^0, \varepsilon, 0) \xrightarrow{dc^*} (q_A, v_A, q_B, v_B, w, T)$ then $\exists C, t_A \bowtie t_B$ such that $(q_A^0, D_A^0, q_B^0, D_B^0, \emptyset, t_A = t_B, \varepsilon, 0) \xrightarrow{ds^*} (q_A, D_A, q_B, D_B, C, t_A \bowtie t_B, w, \lfloor T \rfloor)$ where $(v_A, v_B) \models C$, $v_A \in D_A$, $v_B \in D_B$, and $v_A(t_A) \bowtie v_B(t_B)$.*

Proof. By induction on the length of the path. The symbolic semantics takes transitions which correspond to the concrete ones. Namely, it takes the same discrete transitions. We observe that modifications of clock difference constraints reflect the changes in the fractional parts of the clocks induced by a concrete transition. The detailed proof of this can be found in [KP05a]. Also, all new constraints we add during discrete transitions are consistent with any concrete valuation reached by such a transition.

The only place where updates of $t_A \bowtie t_B$ can violate the lemma are when $t_A < t_B$ changes to $t_A = t_B$ (in case A moves) or when $t_A > t_B$ changes to $t_A = t_B$ (in case B moves), because these transitions are guarded by the predicate *Consistent*. But the concrete valuations reached in the concrete semantics make this predicate true.

Because $T = \text{Sum}(\lfloor T \rfloor, t_A, t_B)$ for any reachable state in concrete semantics, transitions in the symbolic semantics increment or decrement N correctly. ■

Lemma 3 *If $(q_A^0, D_A^0, q_B^0, D_B^0, \emptyset, t_A = t_B, \varepsilon, 0) \xrightarrow{ds^*} (q_A, D_A, q_B, D_B, C, t_A \bowtie t_B, w, N)$ then there is (v_A, v_B) such that $v_A \in D_A, v_B \in D_B$, $(v_A, v_B) \models C$, and $v_A(t_A) \bowtie v_B(t_B)$ and for all such (v_A, v_B) there are $\bar{v}_A \sim_K v_A, \bar{v}_B \sim_K v_B$ such that $(q_A^0, v_A^0, q_B^0, v_B^0, \varepsilon, 0) \xrightarrow{*} (q_A, \bar{v}_A, q_B, \bar{v}_B, w, T)$, where $T = \text{Sum}(N, t_A, t_B)$.*

Proof. By induction on the length of the path (amount of time passed in the concrete desynchronized semantics). The basic step is trivial. For the induction step, we assume that the lemma holds for $(q_A^0, D_A^0, q_B^0, D_B^0, \emptyset, t_A = t_B, \varepsilon, 0) \xrightarrow{ds^*} (q_A, D_A, q_B, D_B, C, t_A \bowtie t_B, w, N)$, i.e., there is (v_A, v_B) such that $(v_A, v_B) \models C$ and $v_A(t_A) \bowtie v_B(t_B)$ and for all such (v_A, v_B) there are $\bar{v}_A \sim_K v_A, \bar{v}_B \sim_K v_B$ such that there is a path $(q_A^0, v_A^0, q_B^0, v_B^0, \varepsilon, 0) \xrightarrow{*} (q_A, \bar{v}_A, q_B, \bar{v}_B, w, T)$, where $T = \text{Sum}(N, t_A, t_B)$. To simplify the notation, we say that v_A, v_B are reachable. Now we consider a transition $(q_A, D_A, q_B, D_B, C, t_A \bowtie t_B, w, N) \xrightarrow{} (q'_A, D'_A, q_B, D_B, C', t_A \bowtie' t_B, w', N')$. If we write (v'_A, v'_B) then we assume that $v'_A \in D'_A, v'_B \in D'_B, (v'_A, v'_B) \models C', v'_A(t_A) \bowtie' v'_B(t_B)$. Note that we use an alternative symbolic path of the same length for the induction hypothesis. We start with moves of A .

Time Pass:

- $D_A \xrightarrow{} D'_A, \exists x \in \text{integral}(D_A), C' = C$
For any (v'_A, v'_B) we take $t = \text{fr}(v'_A(x))$ and we show that $(v'_A - t, v'_B)$ are reachable. Clearly, $(v'_A - t, v'_B) \models C$. To show that $v'_A - t(t_A) \bowtie v'_B(t_B)$ we consider the (in)equalities generated during discrete transitions. Because

there is a zero clock x_B^{zero} in B , we have that $v'_A - t(t_A) - v'_A - t(x) \bowtie v'_B(t_B) - v'_B(x_B^{zero})$ implies $v'_A - t(t_A) \bowtie v'_B(t_B)$. Now there are two possibilities. Either $t_A - x \bowtie t_B - x_B^{zero} \in C'$ which proves the property or $t_A - x \bowtie t_B - x_B^{zero} \notin C'$. But then x was not reset during this uninterrupted series of moves of A , which consists of at least one time pass transition of A . But then the preceding transition was from the group of rules induced by $D_A \longrightarrow D'_A, \exists x \in integral(D'_A)$. Since $t_A - x \bowtie t_B - x_B^{zero} \notin C'$, also other possibilities, namely $(D_A, D_B, C, t_A < t_B, w, N) \longrightarrow_{ds} (D'_A, D_B, C', t_A < t_B, w, N)$ and $(D_A, D_B, C, t_A < t_B, w, N) \longrightarrow_{ds} (D_A, D_B, C, t_A = t_B, w, N)$ from the group of rules induced by $D'_A = D_A, \nexists x \in integral(D_A)$, were enabled. Then, from the induction hypothesis, $(v'_A - t, v'_B)$ is reachable (by a path with different time pass transitions).

- $D'_A = D_A, \nexists x \in integral(D_A), C' = C$

If \bowtie is $<$ then for any (v'_A, v'_B) we take t such that $v'_A - t \in D_A$. Because $Consistent(D'_A, D_B, C', t_A = t_B)$ is true, there is such (v'_A, v'_B) . Also, $(v'_A - t, v'_B) \models C$ and $v'_A - t(t_A) < v'_B(t_B)$.

If \bowtie is $=$ then for any (v'_A, v'_B) we take t such that $v'_A - t(t_A) = v'_B(t_B)$. We need to show that $v'_A - t \in D_A$. Assume that it is not the case and a violating clock is a clock x . Then $t_A - x \bowtie t_B - x_B^{zero} \notin C'$. But then x was not reset during this uninterrupted series of moves of A , which consists of at least three time pass transitions of A . But then a preceding transition was from the group of rules induced by $D_A \longrightarrow D'_A, \exists x \in integral(D'_A)$. Since $t_A - x \bowtie t_B - x_B^{zero} \notin C'$, also other possibilities, namely $(D_A, D_B, C, t_A < t_B, w, N) \longrightarrow_{ds} (D'_A, D_B, C', t_A = t_B, w, N)$ and $(D_A, D_B, C, t_A < t_B, w, N) \longrightarrow_{ds} (D_A, D_B, C, t_A = t_B, w, N)$ from the group of rules induced by $D'_A = D_A, \nexists x \in integral(D_A)$, were enabled. Then, from the induction hypothesis, $(v'_A - t, v'_B)$ is reachable (by a path with different time pass transitions).

- $D_A \longrightarrow D'_A, \exists x \in integral(D'_A), C'$ is computed according to Table IV.3 and Table IV.4. A proof of the correctness of the CDR updates can be found in [KP05a]. For any (v'_A, v'_B) we take t such that $v'_A - t \in D_A$. Because $Consistent(D'_A, D_B, C', t_A = t_B)$ is true, there is such (v'_A, v'_B) . Also, $(v'_A - t, v'_B) \models C$ (update from C to C' ensures this) and $v'_A - t(t_A) < v'_B(t_B)$.

Discrete Transition: $(q_A, D_A) \longrightarrow (q'_A, D'_A), x$ is reset

For any (v'_A, v'_B) we need to show that there is a t such that (v_A, v_B) are reachable, where $v_A(y) = v'_A(y), y \neq x, v_A(x) = t$, and $v_B = v'_B$. Assume that there is no such t .

One cause of this can be an unsatisfiable pair of (in)equalities in C which involve x , i.e., $(v_A, v_B) \not\models C$ for all t . But if we consider any two (in)equalities of the form $t_A - x \leq exp_1$ and $t_A - x \geq exp_2$ then $exp_1 \geq exp_2$ is equivalent to an (in)equality given by the region D_B . It is enough to consider only generation of new (in)equalities to check this, because the CDR updates during the time pass preserve this property. Therefore, a (v'_A, v'_B) for which there is no t with the desirable property does not satisfy $v'_B \in D_B$.

The other cause can be that $v_A \notin D_A$ for all t and $(v_A, v_B) \models C$. This can happen when there are two (in)equalities involving x , one from C and one from

D_A , such that when we project out x then we get another constraint which is not satisfied by (v'_A, v'_B) . Let us denote such constraints $t_A - x \bowtie t_B - y \in C$ and $x - z \bowtie' 0 \in D_A$, the unsatisfied constraint is $t_A - z \bowtie'' t_B - y$ ($(v'_A, v'_B) \not\models t_A - z \bowtie'' t_B - y$). If $t_A - x \bowtie t_B - y \in C$ was introduced by reset of y then $t_A - z \bowtie'' t_B - y \in C$, because newly generated (in)equalities are closed under projection with region constraints. But then also $t_A - z \bowtie'' t_B - y \in C'$.

If $t_A - x \bowtie t_B - y$ was introduced by reset of x then there are two possibilities. Either $t_A - z \bowtie t_B - y \in C'$ (introduced by an earlier reset of z) which solves the problem or $t_A - z \bowtie t_B - y \notin C'$. But then z was not reset during this uninterrupted series of moves of A , which consists of at least one time pass transition of A . But then a preceding transition was from the group of rules induced by $D_A \rightarrow D'_A, \exists x \in \text{integral}(D'_A)$. Since $t_A - z \bowtie t_B - y \notin C'$, also other possibilities, namely $(D_A, D_B, C, t_A < t_B, w, N) \xrightarrow{ds} (D'_A, D_B, C', t_A = t_B, w, N)$ and $(D_A, D_B, C, t_A < t_B, w, N) \xrightarrow{ds} (D_A, D_B, C, t_A = t_B, w, N)$ from the group of rules induced by $D'_A = D_A, \nexists x \in \text{integral}(D_A)$, were enabled. Then, from the induction hypothesis, $(v'_A - t, v'_B)$ is reachable (by a path with different time pass transitions).

The arguments for the transitions induced by B are similar, the only difference is with the conditions on reading. Their correctness follows from the fact that $T = \text{Sum}(N, t_A, t_B)$. Also note that the discrete transitions taken by desynchronized symbolic semantics are the same as those taken by desynchronized concrete semantics along each path. ■

Lemma 4 For a given CTA S , $L_{DS}(S) = L_{DC}(S)$.

Proof. Follows from the proofs of Lemma 2 and Lemma 3, namely the fact that the discrete transitions are the same along any two corresponding paths. ■

Observe that there are only finitely many different clock difference relations over a fixed set of clocks. Then there are only finitely many different symbolic states when we project out N . Since N contains an integer, this system can be replaced by a one-counter machine accepting the same language (actually, a bisimilar one-counter machine).

Theorem 5 State reachability and channel boundedness problems are decidable for CTA of the form $(A_1, A_2, c_{1,2})$.

Proof. Follows from Lemma 1, Lemma 4, and basic language theory. ■

Now we show that the instructions of a one-counter machine can be encoded in a CTA with one channel. The counter is encoded as the number of a 's in the channel. Figure IV.4 shows how to encode incrementation of the counter q_i : $C := C + 1$; goto q_j and conditional decrementation of the counter q_i : if $C = 0$ then goto q_j else $C := C - 1$; goto q_k . Each transition takes exactly one time unit. We omit clocks and guards on all other edges (they are labeled by $x = 1, x := 0$). Test for zero is performed by a nondeterministic choice for

A. To check that the choice was correct, A produces b . If it was wrong then b is not consumed by the corresponding transition of B , stays in the channel and eventually blocks the computation of B . At the end of the computation, B has to check whether there is any b in the channel. If it is the case then it moves to the error location. Note that we cannot use such simple encoding for two-counter machine and CTA with two channels.

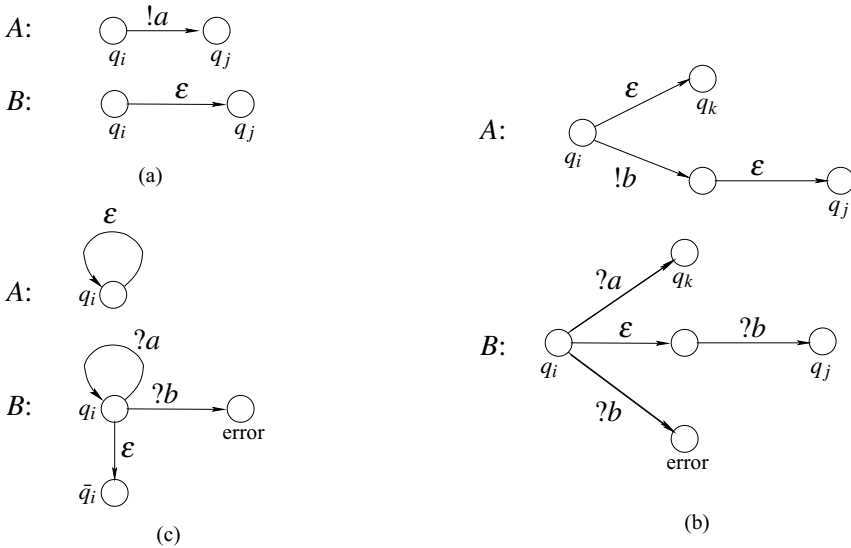


Figure IV.4: A schematic description of a CTA encoding instructions of a one-counter machine. (a) encodes incrementation of the counter, (b) encodes conditional decrementation of the counter, and (c) encodes the final check of the content of the channel.

To illustrate the expressive power of CTA, Figure IV.5 shows a (schematic description of a) CTA which accepts a non-regular context-free language $a^n b a^n b$. Again, each transition takes exactly one time unit and we omit $x = 1, x := 0$ from all edges. The number of a 's is remembered in the size of the channel content and we use different speed of production/consumption to maintain the correct number of a 's in the channel. At the beginning, A produces twice faster than B reads. There are $n/2$ a 's in the channel when B reads the first b and from this moment B reads twice faster than A produces.

From the point of view of the desynchronized semantics, the number of a 's in the channel corresponds to the level of desynchronization. After reading the first n letters a the lag of B is $2n$ time units. Then it reads a dividing letter b and reads a 's again. If there are n letters a then A and B get synchronized again and the accepting configuration is reachable after two more steps. If there are more a 's then B gets stuck reading them, because it reads faster than A produces. If there are less a 's then B can read b immediately and it has to go down to the error state. All locations of A are accepting, but the only accepting location of B is the next to the last one.

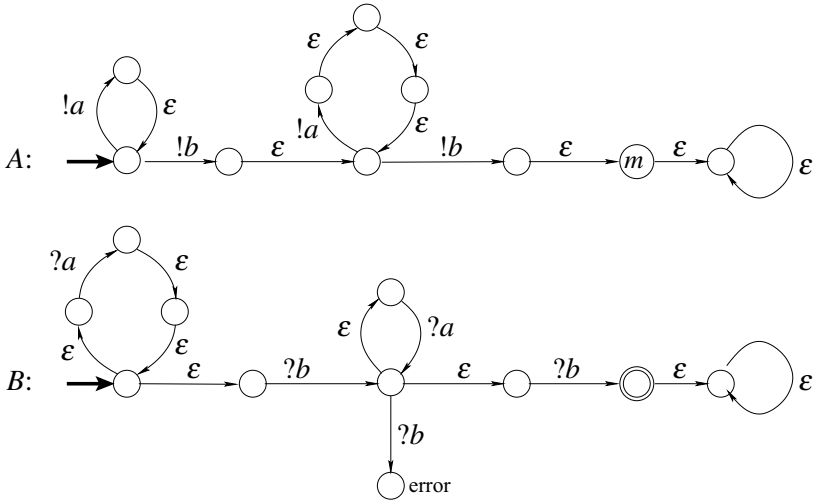


Figure IV.5: A CTA accepting the language $a^n b a^n b$.

This automaton accepts the same language also in discrete time. It also shows the expressive power of CTA with one channel without urgency in the semantics, i.e., ϵ -transitions of B are not restricted. The language accepted by the CTA in Figure IV.5 remains the same even for non-urgent semantics when the only accepting location of A is the location m .

4 CTA with Two Channels

Now we consider systems of the form $(A_1, A_2, A_3, c_{1,2}, c_{2,3})$ shown in Figure IV.2. We show that such CTA have the Turing power. This contrasts with the CFSMs, where systems of this form can accept only regular languages. The notion of the global time changes substantially the expressive power.

We cannot encode counters in the number of a 's as we did it for one-counter machine, because there is no way how to verify nondeterministic choice of A_1 when deciding whether $c_{2,3}$ is empty. We will build on the construction from Figure IV.5. Again, we use different speed of production/consumption to maintain number of a 's in the channels.

To show the simulation of a two-counter machine by a CTA with two channels we first notice that there is a system which accepts a language $a^n b (a^n b a^n b)^*$. Therefore, there is a system which can keep the number of a 's at the same level during the whole computation. It works on the same principle as the system from Figure IV.5. Using the first channel ($c_{1,2}$) and the desynchronization of the automata we check that $2i$ -th and $2i + 1$ -th sequence of a 's have the same length and, at the same time, send the $2i + 1$ -th sequence to the second channel ($c_{2,3}$). Then the same construction is used to

check that $2i + 1$ -th sequence has the same length as the $2i + 2$ -th sequence. A schematic description of this CTA is in Figure IV.6.

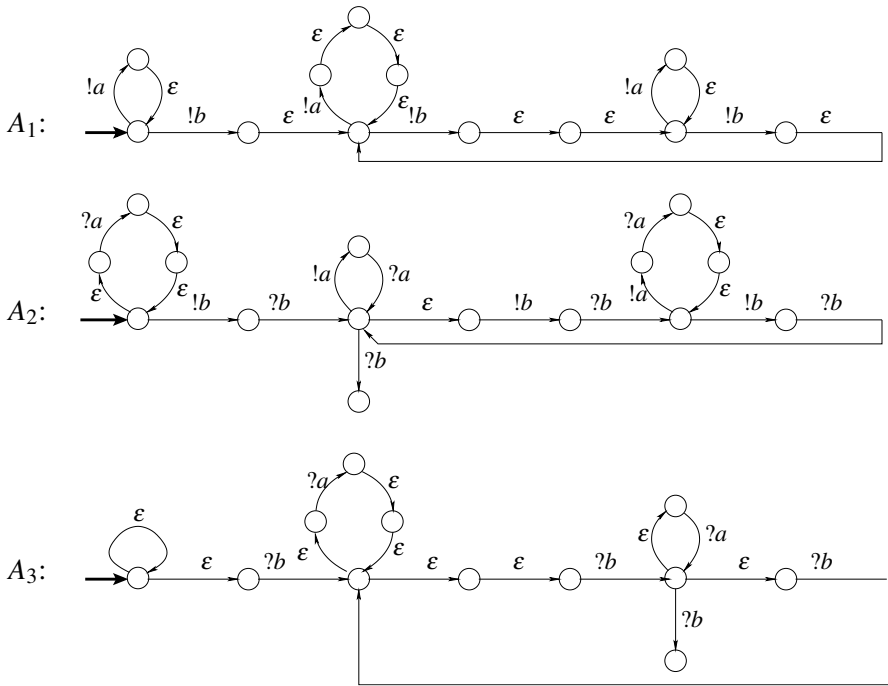


Figure IV.6: A CTA for $a^n b (a^n b a^n b)^*$.

The CTA simulating a two-counter machine accepts a language corresponding to the sequence of the encoded values of the counters during the computation of this two-counter machine. The values m, n of the two counters C_1, C_2 are encoded by the length of the sequence of a 's – the corresponding sequence is $a^{2^n 3^m}$. Therefore, incrementation of the counter C_1 corresponds to doubling of the length of the sequence, decrementation of C_1 to halving, incrementation of C_2 to multiplying by 3, and decrementation of C_2 to dividing by 3. To test a counter for zero, we need to check whether the length of the sequence is divisible by 2 or 3. Basically, we use the same trick as for the language $a^n \text{square}(a^n b a^n b)^*$. Just the consecutive sequences can be of the form $a^n b a^{2n}$, $a^n b a^{3n}$, $a^{2n} b a^n$, or $a^{3n} b a^n$. This can be easily done, since each of these pairs are context-free languages, and the correct overlapping is secured by using both channels in an alternating manner.

Figure IV.7 depicts a fragment for doubling of the length of the sequence of a 's, which corresponds to the incrementation of C_1 . The relative speed of production and consumption is set so that A_2 does not end in the error sink only if the second sequence is twice as long as the first one. The third sequence is as long as the second one (otherwise, A_3 ends up in the error sink), but A_2 gets desynchronized at the same time. This is a preparation for the next operation. Therefore, the simulation of the next instruction does not start with

the first loop, but it goes directly to the second loop (behind the dashed line). This is to ensure overlapping of the length checking. Each transition takes one time unit, we omit guards and resets ($x = 1, x := 0$). All these constructions also work in the discrete time, but they do not work for non-urgent semantics.

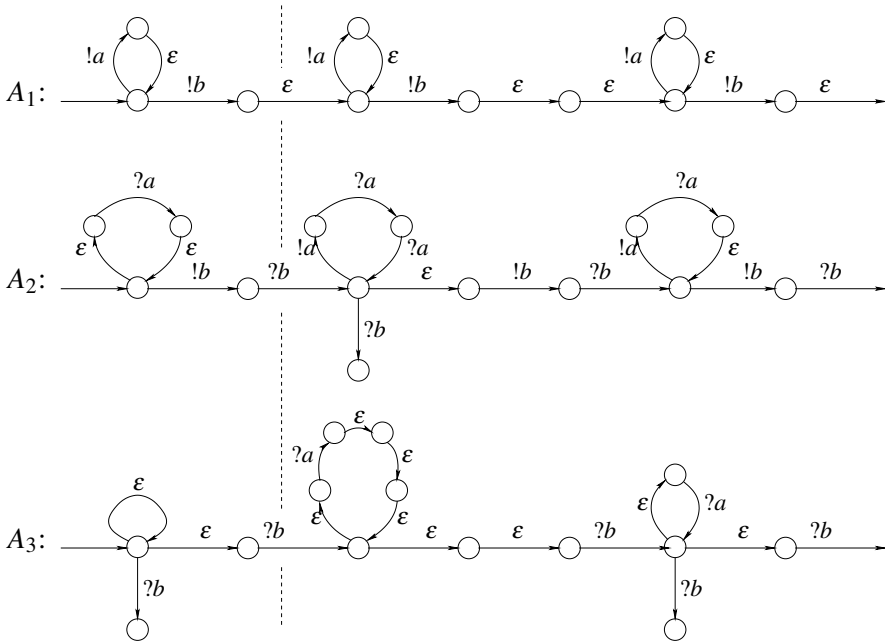


Figure IV.7: A widget for doubling of the number of a 's – incrementation of C_1 .

The halving fragment for decrementation of C_1 is shown in Figure IV.8. Again, when we come to this fragment from some other instruction, we enter behind the dashed line. The first loop corresponds to what is happening in the previous instruction.

Figure IV.9 contains a fragment for the test for zero of C_1 . We need to check whether the length of a sequence of a 's is odd ($C_1 = 0$) or even ($C_1 > 0$). To do this we keep the same length of the sequence and count the number of a 's modulo 2 in the middle loop. The machine cannot cheat, because then A_2 or A_3 would end in a sink state. Therefore, we will leave the loop through a correct branch in all automata.

We do not show fragments for the simulation of the operations on the counter C_2 , but the only difference from the presented fragments is the length of the loops – we need to multiply and divide by three. The halting instruction corresponds to an accepting sink.

Theorem 6 *Reachability for networks of communicating timed automata of the form $(A_1, A_2, A_3, c_{1,2}, c_{2,3})$ is undecidable.*

Proof. Follows from the construction simulating a two-counter machine given above.

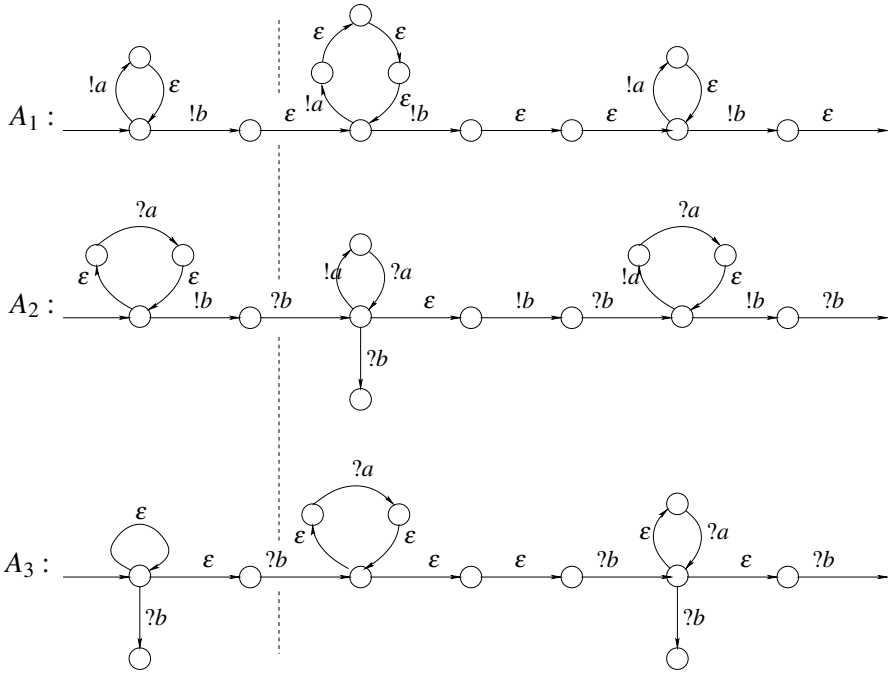


Figure IV.8: A widget for halving of the number of a 's-decrementation of C_1 .

5 Conclusions

To our best knowledge, this is the first attempt to study channel systems in the timed setting. We have proposed CTA as a general framework for modeling of channel systems in which the relative speeds of message production and consumption by local components must meet given timing constraints. Our goal is to mark the basic ground by identifying decidable and undecidable problems for such systems and raises relevant questions for future work. Our technical results may be summarized as follows: (1) CTA with one channel without sharing states in the form $(A_1, A_2, c_{1,2})$ (as shown in Figure IV.1) is equivalent to one-counter machine and therefore questions such as state reachability and channel boundedness are decidable for such systems, and (2) CTA with two channels without sharing states in the form $(A_1, A_2, A_3 c_{1,2}, c_{2,3})$ (as shown in Figure IV.2) has the power of Turing machines. We note that in the untimed setting, channel systems in these configurations are no more expressive than regular languages. It is surprising that the feature of synchronizing on global time makes it substantially more difficult to verify channel systems.

An interesting question related to the timed setting is whether one can synthesize (or modify) the clock constraints of a CTA under given liveness requirements such that the channel is bounded. Another natural direction for future work is to study the complexity of the decidable problems for CTAs. A question following from the previous results on CFSMs [PP92] is the expressiveness of cyclic CTA with one-type messages.

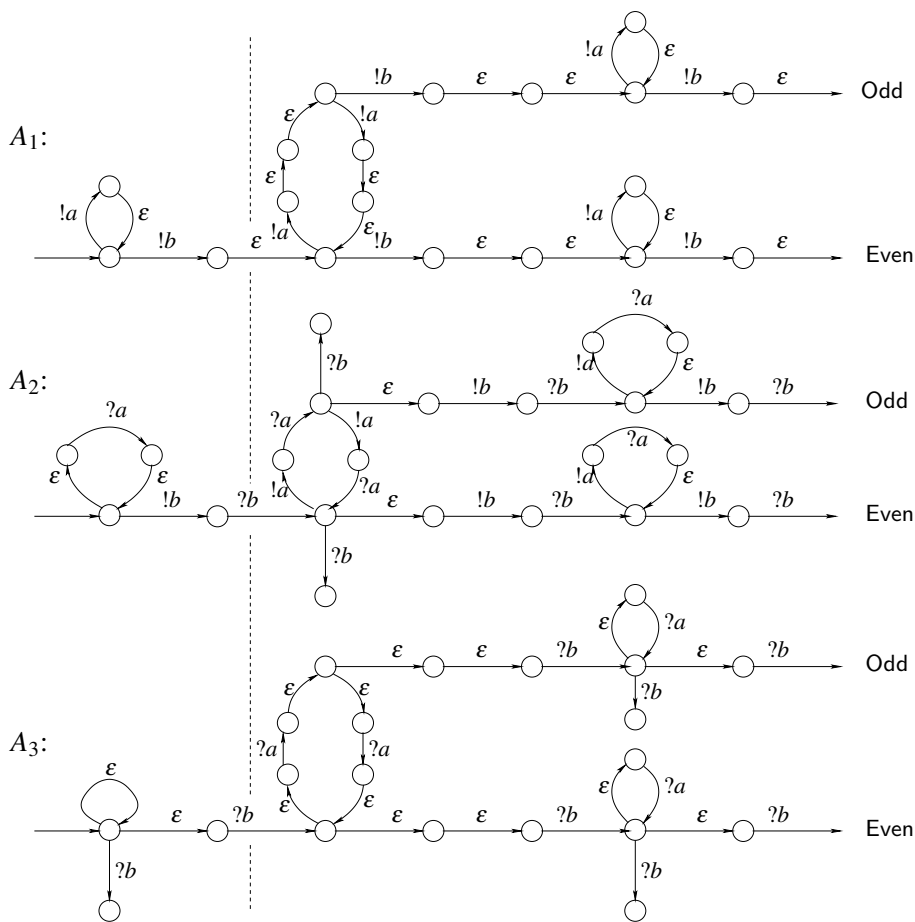


Figure IV.9: A widget for test for odd/even number of a 's - test for zero on C_1 .

References

- [AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
- [AJ96a] Parosh Aziz Abdulla and Bengt Jonsson. Undecidable verification problems for programs with unreliable channels. *Information and Computation*, 130(1):71–90, 1996.
- [AJ96b] Parosh Aziz Abdulla and Bengt Jonsson. Verifying programs with unreliable channels. *Information and Computation*, 127(2):91–101, 1996.
- [BJLY98] Johan Bengtsson, Bengt Jonsson, Johan Lilius, and Wang Yi. Partial order reductions for timed systems. In *Proc. of CONCUR'98*, volume 1466 of *LNCS*, pages 485–500. Springer-Verlag, 1998.
- [BZ83] Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, 1983.
- [CF] Gérard Cécé and Alain Finkel. *Information and Computation*, page 166.
- [CFP96] Gérard Cécé, Alain Finkel, and S. Purushothaman Iyer. Unreliable channels are easier to verify than perfect channels. *Information and Computation*, 124(1):20–31, January 1996.
- [FM97] Alain Finkel and Pierre McKenzie. Verifying identical communicating processes is undecidable. *Theoretical Computer Science*, 174(1-2):217–230, March 1997.
- [FPS03] Alain Finkel, S. Purushothaman Iyer, and Grégoire Sutre. Well-abstracted transition systems: Application to FIFO automata. *Information and Computation*, 181(1):1–31, February 2003.
- [GMK04] B. Genest, A. Muscholl, and D. Kuske. A kleene theorem for a class of communicating automata with effective algorithms. In *Proc. of DTL'04*, volume 3340 of *LNCS*, pages 30–48. Springer, 2004.
- [KP05a] Pavel Krčál and Radek Pelánek. Reachability relations and sampled semantics of timed systems. Technical Report FIMU-RS-2005-09, Faculty of Informatics, Masaryk University Brno, December 2005.
- [KP05b] Pavel Krčál and Radek Pelánek. On sampled semantics of timed systems. In *Proc. of FSTTCS'05*, volume 3821 of *LNCS*, pages 310–321. Springer-Verlag, 2005.
- [Pac82] Jan K. Pahl. Reachability problems for communicating finite state machines. Technical Report CS-82-12, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, May 1982.
- [Pac03] Jan K. Pahl. Reachability problems for communicating finite state machines. *CoRR*, cs.LO/0306121, 2003.
- [PP92] Wuxu Peng and S. Purushothaman Iyer. Analysis of a class of communicating finite state machines. *Acta Informatica*, 29(6/7):422–499, Nov 1992.