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

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 *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.

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: Vítek 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*. Dena 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*. Automatateori 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 automat 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 ökningar 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

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 extend 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.

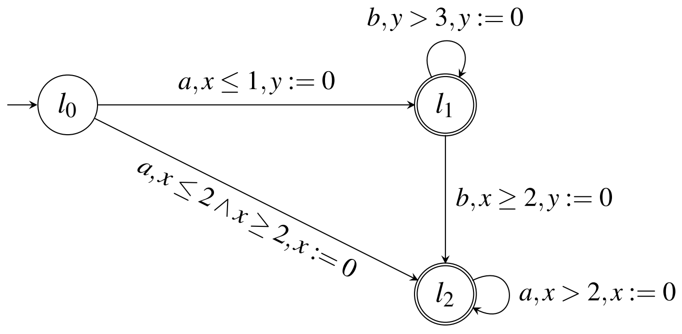

Figure 1.1: An example timed automaton with states l_0, l_1, l_2 and clocks x, y. The state l_0 is initial and the states l_1, l_2 are accepting. The labels on transitions denote the letter, the clock guard, and the reset. For instance, the transition from l_0 to l_2 is labeled by $a, x \leq 2 \wedge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

[1]Other 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 posses 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 posses 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 allow 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 partially[1] 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

[1]It 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 processor 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 $n \cdot \varepsilon$.

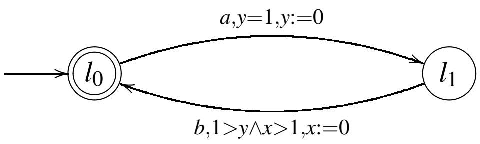

Figure 3.1: A timed automaton which does not preserve qualitative behaviors in sampled semantics. It enforces the difference between clock values to grow. If the values of x, y are $0.1, 0.6$, respectively, in the location l_0 then the difference between the clock values in the location l_0 after reading 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 exits 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 *FTRTFT'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 *FTRTFT'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
UPPSALA
2009