Expressivity of Timed Automata Models

A scientific essay in Natural Science,
Mathematics and Computer Science

Doctoral thesis

to obtain the degree of doctor
from the Radboud University Nijmegen
on the authority of the Rector, Prof. dr. C.W.P.M Blom,
according to the decision of the Council of Deans
to be defended in public on Monday, 11 December 2006
at 10.30 a.m. precisely

by

Biniam Gebremichael Tesfagiorgis
born in Asmara, Eritrea
on 23 August 1971

Expressivity of Timed Automata Models

Een wetenschappelijke proeve op het gebied
van de Natuurwetenschappen, Wiskunde en Informatica

Proefschrift

ter verkrijging van de graad van doctor
aan de Radboud Universiteit Nijmegen
op gezag van de Rector Magnificus
prof. dr. C.W.P.M. Blom
volgens besluit van het College van Decanen
in het openbaar te verdedigen op maandag 11 december 2006
om 10.30 uur precies

door

Biniam Gebremichael Tesfagiorgis
geboren op 23 augustus 1971
te Asmara, Eritrea

to my family

Copyright ©2006 Biniam Gebremichael, Nijmegen The Netherlands
ISBN-10: 90-9021124-1
ISBN-13: 978-90-9021124-4
IPA Dissertation Series 2006-18

Typeset with LaTeX 2$_\varepsilon$
Printed by PrintPartners Ipskamp, Enschede

The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics). It was supported by the European Community Project IST-2001-35304 AMETIST, and by NWO PROGRESS project HaaST.

Contents

Figures and Tables

List of Figures

List of Tables

Preface

I am greatful to many people, who made my PhD studentship a relaxed and successful experience. My supervisor Frits Vaandrager is in the heart of this successful experience. I was the lucky beneficiary from Frits' great knowledge in the area of formal methods and real-time systems. He was my co-author on four papers and proof-reader on the rest of the papers. His intelligent and quite-to-the-point comments on my draft papers have improved the quality of the papers to great extent. I would also like to thank my reading committee: Kim G. Larsen, Holger Hermanns, Howard Bowman, Bart Jacobs and Herman Geuvers for their effort and valuable comments that improved the quality of the thesis.

My research was funded by the EU project AMETIST. The quarterly meeting of AMETIST is the source of my professional experience and a tour to nice places in Europe. I have learned a lot from the AMETIST members; my co-authors Pedro D'Argenio, Tomas Krilavičius, Yaroslav S. Usenko are few of those whose inspiration was very rewarding. My life in Nijmegen was made easy by continuous help from Mirèse Willems, Maria van Kuppeveld, Hennie Claassen, Dé Reinders and the rest of ITA and SOS group members. Thank you!

My parents, brothers and sisters have supported me through out the years and they helped me out in so many different ways. Special thanks to my best friends: Mewael, Yonas, John, Biniam Kahsu, Emma, Biniam Iyob, Tamrat, Kidane, Ruud Leenders . . . have always been there for me when I needed them.

I would like to reserve my final words of acknowledgment to my wife Milen and my son Eben. Without their love, support and understanding this thesis would not have happened.

Nijmegen 2006

Chapter 1

Introduction

We are all familiar and aware of desktop computers and laptops around us. What we are less aware of are, the invisible computers embedded inside vehicles, health equipment, household devices, and many others. We interact with them continuously in our daily life, and despite their invisibility they are far more numerous than the desktop computers.

On the one hand, the existence of these embedded computer systems (*embedded systems* for short) has greatly improved our living standards. Education, health services, agriculture, ... are now more efficient and safer than they used to be. On the other hand, malfunctioning of these computers has consequences ranging from simple annoying behavior to multi million Euros losses or even death. The destructive failures witnessed in Mars Pathfinder [Jon97], Ariane 5 [J.L96] and the "Friendly Fire" death accident in Afghanistan [Loe02] are some examples triggered by an error inside an embedded system. The challenge facing today's industries is thus, how to ensure the safety and correctness of embedded systems under all possible scenarios within the environment in which they are required to work.

The future seems even more challenging than the present. Nowadays, embedded systems are more or less stand-alone systems. For instance, electronic services inside a car, such as airbag control, parking assistance, temperature control, belt tensioner and cruise control have dedicated computing resources, and they operate separately. But as desktops were once stand-alone and now they are fully networked, embedded systems will soon integrate. As a matter of fact, the integration has already started. This will certainly lead to more complex systems, and consequently the cost of malfunctioning will be unbearable.

1.1 Formal Methods

Formal methods is the applied mathematics of computer system engineering. It is an approach used to formally (mathematically) specify systems in

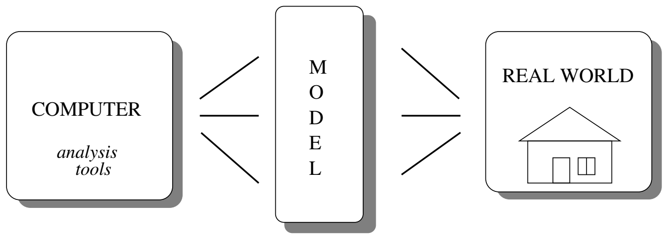

Figure 1.1: Computers, models and real world (from [Smi85])

such a way that relevant properties of the system can be analyzed symbolically prior to production or afterwards.

The application of formal specification and analysis is illustrated in Fig. 1.1. The box on the right-hand side represents real world systems. This may include the environment, machines, controllers, etc. Real world systems are formally specified (*modeled*) in terms of mathematical languages or modeling languages. The model is then rigorously *analyzed* against a relevant property using computer programs represented by the box on the left-hand side.

The ultimate goal, in this model based formal analysis, is to infer relevant properties of the real world based on the results obtained from the analysis of the model. The success of such practices obviously depends on (1) the quality of the model in representing the real world, and (2) feasibility of the analysis of the model. These two requirements are often referred to by the formal methods community as *expressivity* and *computability* of the modeling language in question. A language with more expressive power is often computationally expensive. Conversely, a language which is computationally inexpensive may hardly express real world problem adequately. For a language to be usable, in this sense, it should find an optimal compromise between these two often conflicting requirements. A common approach to optimize expressivity versus computability is to focus on specific problem areas. In this way the expressivity of the language is kept to a minimum in order to optimize computability. One of these special sets of problems is real time system, which is also the main focus of this thesis.

1.2 Real-Time Systems

Real-time systems are systems where correctness is determined by both *temporal* and *functional* correctness of the system. Typical examples of real time systems include train gate supervision, air-traffic control, and networked multimedia systems. Real-time systems are usually embedded inside bigger systems that interact with their environment according to strict timing requirements. Train gate supervision, for instance, is part of train-road safety system that is responsible for closing and opening the gate during the time when the train crosses a multi-user road.

Due to industrial competition, the need to increase performance and share resource among components, the complexity of real-time applications is increasing very fast, which in turn has lead to the introduction of more errors into systems. Apparently the need to formally specify and rigorously verify the correctness and safety of real-time systems is more demanding than ever before.

Formal methods has achieved remarkable success in the VLSI (very large scale integration) industry. Motivated by this, people started to apply formal methods to real-time systems. Many achievements have already been attained in the past two decades in the area of real-time systems verification[1], that have the potential to repeat the same success. These include various theoretical foundations [ACH$^+$95, FZ95, TY01, etc], efficient verification tools [LPY97, BDM$^+$98, BM02, HHWT97, BGO$^+$04, etc] and practical application of formal verification on real-world systems [Feh99, BFK$^+$98, BJMY02, BGK$^+$02, Chap. 6, Chap. 7,etc]. As noted earlier, successful application of formal methods on real-time systems depends on the ability of the proposed formalisms to (1) express real-time systems adequately and intuitively, while at the same time (2) providing an optimal balance between expressivity and computability. The present thesis addresses these issues with a special focus on one type of formalism known as *timed automata* – a popular and widely used framework for real-time systems specification and verification.

The thesis discusses the expressivity of timed automata models with respect to the following two criteria.

1. "*Theoretical expressivity*": One language is said to be (theoretically) more expressive than another language, if whatever the later can express can also be expressed by the former, while the reverse is not necessarily true.

2. "*Practical expressivity*": deals with the ease and naturalness of a language in expressing real-life systems. For instance, a Turing machine is theoretically as expressive as a programming language, however writing a usable program in a Turing machine language is far more complex than in a programming language.

In what follows, we briefly introduce timed automata, and give pointers to related work on the expressivity of timed automata and its variants.

[1]See, for example, [Wan04] for an extensive survey on formal verification of real-time systems.

1.3 Timed Automata

1.3.1 Modeling Time

In real-time system modeling, time can be represented as discrete – where time readings constitute a monotonically increasing sequence of integers [HMP92, Lam05], or as dense – where time readings form a monotonically increasing sequence of reals [AD94, Dil89]. The advantage of discrete-time models is that they can easily be transformed to existing un-timed models; however it is less natural to express physical systems that operate over continuous time as discrete time. Another advantage of continuous-time-models is that they can be represented symbolically (for instanse using difference bound matrices [AD94, BY03]) which is computationally efficient, when compared to discrete-time representations, for many real-time applications.

Alur and Dill's *timed automata* [AD94, ACD90] is one of the first dense-time specification and analysis frameworks for real-time systems. A timed automaton is a finite-state automaton equipped with a finite set of real-valued clocks to measure the progress of time. It is structured as a directed graph whose nodes are *control locations* and whose edges are *transitions*. A transition is augmented with a label, clock comparison (as an enabling condition) and clock resets. Clocks in timed automata can be reset to zero or can be compared to an integer, and between two resets its derivative with respect to time is equal to one. Two interesting results for the computability of timed automata are that reachability analysis [Yov98] and the emptiness problem [AD94] are decidable.

Timed automata is a sensitive language in the sense that a few changes in respect to clock upgrade or the way clocks are compared may easily lead to undecidability of reachability analysis. For instance, stopping clocks for a while and resuming (commonly known as *stop watches* [KPSY99, CL00]) is a useful feature to model preemptive scheduling. However reachability analysis for stopwatch automata is undecidable in general. Nevertheless, timed automata allow some flexibility, provided that the changes are made carefully without violating the decidability results or if they are applied to a selective subclass of timed automata. For example, a more relaxed clock update, which is more convenient than resetting clocks only to zero and maintain decidability, has been proposed in [BDFP00, SV96]. For an extensive discussion of different flavors of timed automata and their decidability results we refer the reader to Alur and Madhusudan's survey [AM04].

Note that, the original timed automata model [AD94, ACD90] allows clocks to be compared only to constant integers (like $x \leq c$, where x is a clock and c is a constant non-negative integer.) People have straightforwardly extended timed automata with constraints like $x_1 + c_1 \leq x_2 + c_2$ [HNSY94]. However Bouyer [Bou03] showed that the model-checking algorithm in [ACD90, HNSY94] is not correct for such an extension.

One of the assumptions of the theory of timed automata is that clocks grow with a constant rate. However in practice, physical clocks are not perfect and they can not maintain a constant rate of growth. Several researchers [ATM05, HVG03, KLSV03a] propose variants of timed automata with perturbed clocks, where clocks are assumed to advance at a rate of $1 \pm \epsilon$ with respect to time. These automata specify real-time systems more accurately, but their behavior can only be analyzed approximately. A more general approach is to allow clocks to progress with their own rate (given as a differential equation w.r.t. time). This generalization of timed automata leads to another class of systems known as *hybrid systems* [Hen96, LSV03, etc].

Another important issue in modeling timing behavior of real-time systems is urgency. It is a common scenario in real-time systems that some actions *must* be taken urgently without any further delay, or before a given deadline expires. There are several approaches to express this scenario within timed-automata-based modeling. Since urgency is one of the main focuses of the present thesis, we give below a detailed discussion of this issue.

1.3.2 Urgency for Timed Automata

The original definition of timed automata [AD94, ACD90] did not have explicit syntax to control the progress of time. Instead a Büchi style acceptance criterion is used by requiring that some (sets of) locations are visited infinitely often. A popular approach, which is advocated in [HNSY94, AH94a, HKWT95] and implemented in the tool UPPAAL [LPY97], is to add invariants in locations. An invariant imposes a *hard deadline* on the system by limiting the amount by which time may advance in a given location/state. A similar approach pursued in [KLSV03a] is to use stopping conditions. Here the idea is that when a system reaches a state in which a stopping condition holds, time may not progress any further and a transition has to occur immediately.

The last two approaches assume synchronization between two components can always be taken before one of the components violates its invariant (or stopping) condition. But not all real-time systems work this way. Take the following as an example. When a customer pays the appropriate fee and presses the coffee button on a coffee machine, a coffee should be served as soon as possible. This may take some time – eg. depending on the temperature of the water – but as soon as all ingredients are ready the coffee should be ready without further delay. Location invariants and stopping conditions are less appropriate for such type of behaviors, because they are not time-reactive (unable to guarantee the existence of an enabled transition, when time progress is blocked [BS00, Bow99].) Expressing this behavior using invariants/stopping conditions may result in unnecessary errors, namely *time deadlock* (a state in which there is not transition enabled and time is forbidden to proceed).

Based on this observation, Sifakis and his colleagues [BS00, SY96] advocate the use of deadlines for the specification of time progress properties. Each transition of a timed automaton [AD94] is decorated with an additional deadline predicate, which specifies when the transition becomes urgent. An advantage of the deadline approach is that under some reasonable assumptions it ensures absense of time deadlock. Timed automata with deadlines are implemented in the IF tool suite [BGO+04], and partially by UPPAAL[BLL+96].

The same approach, with slight modification, is presented in Chapter 4 which advocates urgency predicates instead of deadline predicates. Here deadline predicates are computed as a disjunction of the guard and the urgency predicate. As a result of this computation, the assumption that deadline implies guard is guaranteed by construction. We have applied this approach to Timed I/O Automata [KLSV03a] and it has resulted in shorter specifications and better expressivity (in a practical sense), the "theoretical" expressivity is essentially the same.

Another related work on modeling urgency is by Barbuti and Tesei [BT04]. In this paper they study a different aspect of urgency, namely, the semantics of *as soon as possible* with arbitrary non-zero precision. Their study is based on timed automata with urgent predicates where urgency predicates are assumed to be either true or false[2].

1.3.3 Logic and Algebra for Real-time Systems

On another front, there has been extensive research on extending temporal logic to real-time system modeling. Similar to the idea of extending finite automata with time, temporal logic has also been extended by adding time quantitatively. TPTL [AH94b] and MTL [Koy90] are the two main time extensions of linear temporal logic [MP92, Lam83]. Interested readers may refer to [MNP05, Hen98, Ras99] for a general survey on timed temporal logic, and [BCM05] for the expressivity difference of MTL and TPTL. A timed version of branching temporal logic known as TCTL also appears in [ACD93] and its model checking algorithms are given in [ACD93, TY01, CE81].

Process algebra models such as CCS [Mil89a], CSP [Hoa78] and ACP [BK82] are also extended with time in [Yi91, NS91, FZ95, BM02]. Similarly Timed Petri nets [Mer74, Ram74] have been proposed as a timed extension of Petri nets. Interested readers may consult [BCH+05] for a comparison between timed automata [AD94] and timed Petri nets [Mer74], which

[2]Note that restricting urgent predicates to true or false does not reduce the expressivity of the model, but may have more transitions than required by the unrestricted model.

shows both languages are equally expressive w.r.t timed language acceptance, while timed automata are more expressive w.r.t weak timed bisimilarity. Similar discussion can also be found in [BG06].

Extension of regular languages into timed regular languages [ACM02, Dim01] is also interesting work that helped in the understanding of timed languages in similar ways to the classical theory of regular languages. Recently, Asarin [Asa04] published open questions and challenges in the area of timed languages to fill the gaps that are left untouched by the research communities which mainly focus on application and analysis tools.

1.4 Overview

The research in this thesis is conducted in the context and under the auspices of the European project – advanced methods for timed systems (AMETIST[3]) carried out during a period of three years and three months (April 1 2002 - June 1 2005). The AMETIST consortium was composed of seven academic and four industrial partners from five different countries. One of the main objectives of the project was *to develop a powerful modeling methodology based on the timed automata model that can express real life situations in complex industrial systems*.

The first part of the thesis (Chapters 2 - 4) compares theoretical expressiveness of timed automata models. The thesis focuses on the issue of time progress in the light of the three most popular timed automata models; namely, Timed Automata with Invariants (called Safety Timed Automata in [HNSY94]), Timed Automata with Deadlines [BS00] and Timed Input/Output Automata [KLSV03a]. The second part of the thesis (Chapters 5 - 7) concerns the practical application of timed automata. These chapters test the expressivity of timed automata against complex industrial case-studies and network protocols proposed in the context of the AMETIST project. The results obtained from the specification and verification of the case studies provide further evidence that timed automata models are mature enough to express complex industrial systems accurately and naturally.

In the reminder of this section we give a brief introduction to each of the chapters.

- Chapter 2 [DG05] presents a new bisimulation relation called *drop bisimulation*. It has been known for several years that the delayability of synchronization of TADs makes strong bisimulation [Mil89a] fail to be a congruence for parallel composition. Chapter 2 solves this problem by characterizing a new bisimulation relation, which is the

[3]`http://ametist.cs.utwente.nl`

coarsest congruence included in the bisimulation relation. An equivalent symbolic bisimulation is also provided, which allows us to prove that drop bisimulation is decidable.

- Chapter 3 [DG06] is a continuation of Chapter 2 and provides an algebraic proof system for direct derivation of equivalence (drop bisimulation) by purely syntactic manipulation. In this chapter we show that the proof system is sound and complete.

- Chapter 4 [GV05] can be viewed as an alternative solution to the delayable synchronization and failing strong bisimulation. The solution is to partition the set of synchronization actions into input and output actions. In this way there is no need for a new bisimulation relation, the strong bisimulation[Mil89a] is compositional by construction. But the primary goal of the chapter is to introduce a new modeling scheme for the time progress condition of TIOA which is shorter and more natural than the earlier stop-when predicate specification. The chapter also presents comparisons and translation schemes among the three ways of specifying time progress.

- Chapter 5 [GKU04] presents the formal modeling and analysis of the Car Periphery Supervision (CPS) system using timed automata. This case study was proposed by one of the AMETIST industrial partners, Robert Bosch GmbH. The CPS system is a central supervision system embedded inside a car. The system collects information from the surrounding environment (For example, the distance and speed of objects driving close to the car) and triggers applications such as air bag inflation, belt tensioner or parking assistant. The CPS system is hybrid in nature, however we made an appropriate abstraction of the environment, which permitted us to model it as a timed automata model. All relevant properties have been verified using the timed automata model checker UPPAAL. The abstractions employed in this case study preserve the sufficient conditions (as stated in [HH95, ACH^{+}95]) that ensures the validity of the verfication results.

- Chapter 6 [GV03a] reports the second case study proposed by another AMETIST industrial partner, Cybernetics. The case study concerns a smart card personalization machine. This machine takes piles of blank smart cards as raw material, programs them with personalized data, prints and tests them. One of the research problems in this case study is to synthesis a controller that can recover the system from an uncontrollable error. Control synthesis tools are not mature enough to handle such a big system ($> 10^{13}$ states). Therefore we model the system using SMV and somehow "hack" SMV (automatically generating SMV models and feed them to the verification engine) to generate controllers that can recover the system from an erroneous state. The work illustrates how a model checker can be used to synthesize a controller that recovers the system from an uncontrollable error.

The reader may be surprised to see a case study concerning the application of SMV, a model checker for (untimed) finite state machines, within a thesis which for the rest is devoted to timed automata. Within the research projects AMETIST and HaaST, by which this PhD research was funded, the academic partners have been assigned the task to solve practical problems proposed by the industrial partners. It has been said that to a hammer everything looks like a nail, but we believe that industrial problems should be tackled by the most appropriate tool that is available. For the Cybernetics controller synthesis problem, a discrete time model appeared to be most natural, and the size of the state space required the use of a discrete time model checker such as SMV. We have included this case study here since we think it is a nice application of model checking techniques. Elsewhere, Uppaal has been used to address different aspects of the Cybernetics case study (see [Mad04]) and we believe it is a nice challenge for timed automata tools to deal with these different aspects in an integrated manner.

- Chapter 7 [GVZ06a, GVZ06b] models and analyzes the Zeroconf protocol. This case study has been proposed by Philips in the context of another research project HaaST[4]. The Zeroconf protocol allows the configuration of IPv4 without the need for a third party server such as DHCP. We decided to model the system in two phases. The first phase makes a more realistic model of the system without taking consideration of the complexity of the model. In the second phase we made a careful abstraction of the model. It is constructed in such a way that automatic verification is possible in UPPAAL. Every abstraction step either preserves equivalence (modulo bisimulation) or is a result of an explicitly stated assumption. The contribution of this chapter is two fold. Firstly, it provides a formal specification of the most critical part of the protocol. Our modeling efforts revealed several errors (or at least ambiguities) in the protocol standard that no one has spotted before. Secondly, relevant properties are verified against the specification that reveals what properties are satisfied under what assumption.

[4]HaaST (Verification of Hard and Softly Timed Systems) is a PROGRESS project `http://fmt.cs.utwente.nl/HaaST/`

1.5 Conclusion

The state space explosion problem was and still is the main challenge for the scalability of formal verification. This is a problem that occurs due to the huge number of states of large systems [BCM$^+$92]. In recent years, there have been significant improvements in tackling the problem. These improvements are not only due to the advancement of computer hardware which doubles every 18 months (Moore's law), but also due to algorithmic techniques, such as abstraction, symbolic analysis and compact data structures. The thesis demonstrates the increasing power of model checkers in Chapters 5, 6 and 7. It uses state-of-the-art modeling and verification techniques to verify the desired properties. This would not have been possible if it was not for the newly devised algorithmic techniques.

However, there are still many cases in which exhaustive space exploration is not feasible. A different and promising approach to this problem is compositional analysis. Efficient implementation of compositional analysis will still depend on clever algorithmic techniques, but the key success comes from the expressivity power of the modeling languages in specifying components separately with clearly defined interfaces, and a means to infer properties of the system from its components.

Component-based analysis is widely used in VLSI industries, but for real-time systems, it has not been exploited thoroughly. This is partly due to the difficulty in reasoning about systems compositionally in the presence of global time. The timing behavior of a component in real time system depends on the global time which can be manipulated (in general) by every component of the system.

A small but significant achievement toward compositional reasoning of real time system is the approach of [BGS00, BS00, Bow99, BGS05] that guarantees the absence of time deadlock and zenoness by construction. The contribution of the thesis in this regard is in providing a means to substitute equivalent components within a system (see Chapter 2). This is an important tool for component-based analysis of real-time systems, because it allows a component to be replaced with an equivalent component without affecting the parent system. Computing such equivalence is decidable and an algebraic theory that allows syntactic derivation of equivalence is given in Chapter 3.

There is still a long way to go until compositional analysis for real-time systems is fully exploited, which could be an interesting research direction for future. In the mean time, for systems that can not be verified automatically, the specification part of formal methods alone can be of great assistance in improving the quality of the system via simulation and documentation. Formal specification can reveal inconsistencies and ambiguity in the early design process, and forces designers to clarify ambiguities and overlooked details. For example, the numerous computer protocols that have been standardized by IEEE SA and other standardization bodies are written in natural languages. These documents will obviously inherit the ambiguity that is present in the natural languages. There are several reports in the literature that demonstrate the usefulness of formal specification in improving the quality of complex design documentation (atleast for the critical part of a system). The thesis (see Chapter 7) provides further evidence that formal specifications are indeed useful in this regard, some formal specification and verification tools (in particular UPPAAL) have reached a level where they can be used for this task. As a side effect, the thesis proposes a number of improvements and additional features to make specification languages (in particular UPPAAL) more "practically" expressive.

Chapter 2

Bisimulation Relation for Timed Automata with Deadlines

with Pedro R. D'Argenio

Abstract Delaying synchronization of actions may reveal some hidden behavior that would not happen if the synchronization met the specified deadlines. This precise phenomenon makes bisimulation fail to be a congruence for the parallel composition of *timed automata with deadlines* – a variant of timed automata where component synchronization is delayable, and time progress is controlled by deadlines on transitions instead of invariants on locations. This problem has been known and unsolved for several years. In this chapter we give a characterization of the coarsest congruence that is included in the bisimulation relation. In addition, a symbolic characterization of such a relation is provided and shown to be decidable.

2.1 Introduction

Design and specification languages allow to model systems in a modular manner by linking small modules or components using the language operations —such as sequential composition or parallel composition— in order to build larger modules. Hence a desirable requirement is that the language is *compositional* with respect to its semantics. By compositional we mean that components can be replaced by behaviorally equivalent components without changing the properties of the larger model in which they are embedded. The preservation of such properties can be guaranteed by means of semantic equivalences or preorders. For example branching bisimulation preserves CTL* [BCG88], language inclusion preserves

LTL [Lam83] and, in particular, timed bisimulation preserves (timed) properties expressed in logics such as TCTL [TY01]. Hence, compositionality amounts to requiring that relations like these are *congruences* (or precongruences) for the different operations of the language.

Timed automata [AD94, HNSY94] are used to model real-time systems and have become popular as modeling language for several model checkers because of their simplicity and tractability [BDL$^+$01, BDM$^+$98, BGO$^+$04]. Timed automata are automata with the additional ingredients of *clocks*. Clocks are variables that increase at the same rate in order to register time progress. Transitions are labeled with constraints on clocks, called *guards*, that indicate when such transition *may* take place. Usually timed automata are used to model real-time systems with *hard* constraints. In this case, timed automata are equipped with an *invariant*, which is a constraint on clocks that limits time progress in each control state [HNSY94]. For a sytem composed of several components time can progress in the system if all its components respect their time progress constraint. The problem with such strong requirments on time progress is that they can easily introduce *time deadlock* (the composed system has reached a state where time is blocked forever [BS00, Bow99, Bow01]). For instance, when two components are in a state from which they will never synchronize, it may be desirable not to further constraint time progress.

The second problem with invariants is that, there is no way to insure absence of time deadlock in a system from its components. Note that time deadlock in timed automata is a serious issue because: (1) it is a generic problem, that is, if an independent component is composed with a component that is time-deadlocked, then the composed system inherits the time deadlock; and (2) the verification of many properties explicitly depend on the absence of time deadlock.

A variant of timed automata called *Timed Automata with Deadlines* (TADs for short) is proposed by Bornot and Sifakis [BS00, SY96, GS05a] to address the above issues. The work in [BS00, SY96, GS05a] shows how time-deadlock-freedom of a system can be inferred from its components, in such a way that, if all components satisfy some *time-deadlock-freedom condition* then it is guaranteed that the system is time-deadlock-free by construction. This model is nowadays embedded in modeling languages such as IF [BGO$^+$04] and MoDeST [DHKK01, BDHK04], and urgent transitions in UPPAAL [BLL$^+$96] can be seen as a particular instance of TAD transitions.

TADs do not have invariants. Instead, a TAD transition has associated a second clock constraint, called a *deadline*, that indicates in which moment such a transition *must* be taken. As a consequence, a deadline is required to hold *only if* the corresponding guard holds ensuring the transition can be taken after the deadline is reached. In this sense, the deadline imposes an *urgency constraint*.

Contrary to the traditional timed automata setting, bisimulation in the

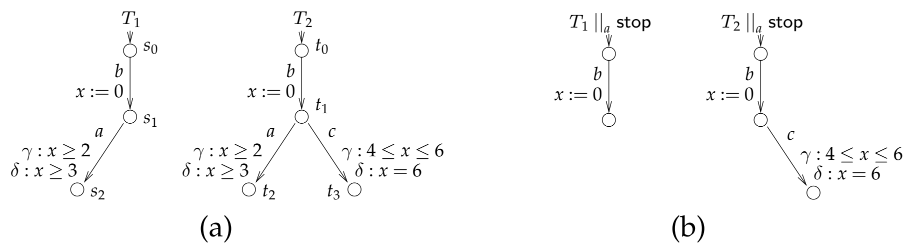

Figure 2.1: TAD and compositionality

TAD model *is not* preserved by parallel composition [BS00]. This is illustrated in the following example. T_1 in Fig. 2.1.(a) depicts a TAD in which circles represent control state and arrows are control transitions. In particular the small incoming arrow identifies the initial state. T_1 performs first an action b at any moment and sets clock x to 0. As time progresses, the value of x increases and when it takes value 2, action a becomes enabled. This is controlled by guard $\gamma : x \geq 2$. At any point after x takes value 2, this transition may take place, but as time continues to progress and x takes value 3, the deadline $\delta : x \geq 3$ obliges the execution of the transition. Notice that T_2 shows a similar behavior since action c cannot be executed: the deadline of a obliges its execution before the guard of c becomes enabled. In fact, T_1 and T_2 are timed bisimilar in the sense of [BS00].

Suppose now that T_1 is composed in parallel with the automaton stop requiring synchronization on action a. (stop is the automaton with a single location and no transition; hence, it does not do anything but idle.) This blocks the execution of action a in T_1. The resulting automaton $T_1 \|_a$ stop is depicted in Fig. 2.1.(b). Similarly, the composition of T_2 with stop in $T_2 \|_a$ stop also blocks the execution of a, but in this case time progresses beyond 3 time units allowing the execution of c after 4 time units (see Fig. 2.1.(b)). As a consequence $T_1 \|_a$ stop and $T_2 \|_a$ stop are not bisimilar.

To the best of our knowledge there is no characterization of a congruence for parallel composition on TADs. The only exception is what is called strong congruence in [BS00], which is the usual bisimulation applied directly on TADs. This relation is, however, far too strong as it requires the syntactic equality of guards, deadlines, and clock resets.

In this chapter we present a congruence relation for parallel composition and prove that it is the coarsest congruence included in the bisimulation relation. This new relation, which we call ∇-bisimulation (read "drop-bisimulation"), is in fact the usual bisimulation on an extended semantics of TAD. Such semantics allows for time progressing beyond deadlines but carefully accounting for the actions whose deadline have been overruled. We also give a symbolic characterization of ∇-bisimulation, that is, a relation defined directly on TADs. As a corollary of this characterization, we obtain that ∇-bisimulation is decidable. Another particular contribution of this chapter is that the proof of congruence is entirely carried out at a symbolic level (i.e., without resorting to the underlying transition system in which ∇-bisimulation is defined). We also discuss different kinds of parallel compositions on TADs (mostly defined already in the literature) reporting which of them preserves ∇-bisimulation and which do not and why.

Related Work.

The failure of bisimulation to be a congruence becomes apparent when soft deadlines are considered, that is actions that may be urgent in isolation are required to wait if they are intended for synchronization i.e. synchronizing actions need to be *patient*. This problem has appeared in the context of stochastic process algebra where synchronization is required to be patient (e.g. [Hil96, Her02, D'A99]). It becomes evident (in a similar manner as above) if bisimulation is considered for the underlying probabilistic transition system rather than for the finer symbolic model [D'A99]. The problem of compositionality also showed up in other process algebras for performance behavior [Cor98].

In [JLS00], compositionality is studied on timed automata with urgent actions w.r.t. simulation. (An urgent action corresponds to an action in TADs for which guard and deadline are the same.) In this case, it suffices to add a condition of readiness on the urgent actions to achieve precongruence. Recently, [GV05] defined a variant of TADs where actions are distinguished between input and output following the model of [SGSAL98] and for which bisimulation *is* a congruence for the parallel composition. This is possible due to input enabling and to the fact that only output actions are allowed to be urgent (i.e. to have deadlines.) Therefore there is no need to wait for synchronization as it is always possible. Though the restrictions imposed by [GV05] makes the new model much simpler and tractable, using it to describe soft real-time systems may result in complex models.

In addition to the solution for the compositionality problem, we also give a symbolic characterization of the congruence. Our work is based on the result of Lin & Yi [LY02] who gave a symbolic characterization of the bisimulation for timed automata. In turn, their result is based on Čerāns' who determined that bisimulation for timed automata is decidable [Čerāns92]. We use also this result to show the decidability of ∇-bisimulation.

Outline

The chapter is organized as follows. Section 2.2 gives the preliminaries recalling timed automata with deadlines, its semantics in terms of transition systems, the definition of bisimulation, and particularly, the definition of parallel composition. In Section 2.3 we discuss the pitfalls of the composition and progressively construct the semantics that leads to the definition of ∇-bisimulation. The symbolic characterization is provided in Section 2.4 and shown to be the coarsest congruence in Section 2.5. Section 2.6 discuses decidability of ∇-bisimulation and the different kind of synchronization in parallel composition.

2.2 Preliminaries

Timed Automata with Deadlines.

A *clock* is a non-negative real-valued variable, which can be reset to zero at the occurrence of an event, and between two resets, its derivative with respect to time is equal to 1. We denote $\mathcal{C} = \{x_1, \ldots, x_N\}$ to be a finite set of clocks. A *clock constraint* $\mathcal{F}(\mathcal{C})$ is a conjunction of formula(s) of atomic constraints in the form of $x_i \bowtie n$ or $x_i - x_j \bowtie m$, where x_i and x_j are clocks in $\mathcal{C}$, $\bowtie \in \{<, >, \leq, \geq, =\}$ and n, m are natural numbers. The constraints **tt** and **ff** are used to denote, respectively, the atomic constraints which are constantly true and false. We will assume that there is a global finite set of actions $\mathcal{A}$ for all timed automata with deadlines.

Definition 2.1 *A* timed automaton with deadlines *[BS00] (TAD for short) is a structure $T = (\mathcal{L}, \mathsf{l}^0, \mathcal{C}, \twoheadrightarrow)$ where*

- *$\mathcal{L}$ is a finite set of locations,*
- *$\mathsf{l}^0 \subseteq \mathcal{L}$ is the set of initial locations,*
- *$\mathcal{C}$ is a finite set of clocks,*
- *$\twoheadrightarrow \subseteq \mathcal{L} \times (\mathcal{A} \times \mathcal{F}(\mathcal{C}) \times \mathcal{F}(\mathcal{C}) \times 2^{\mathcal{C}}) \times \mathcal{L}$, is a finite set of edges.*

If $(s, a, \gamma, \delta, \mathbf{x}, s') \in \twoheadrightarrow$ we write $s \xrightarrow{a,\gamma,\delta,\mathbf{x}} s'$ and require that $\delta \Rightarrow \gamma$ holds, moreover we assume δ is left-closed (left-closure is formally defined in Def. 2.5).

The notion $s \xrightarrow{a,\gamma,\delta,\mathbf{x}} s'$ represents an edge from location s to s' that executes action a whenever *guard* γ becomes true. In addition, *deadline* predicate δ imposes an urgency condition: the transition cannot be delayed whenever δ is satisfied. When executing the transition, clocks in $\mathbf{x}$ are set to 0.

Parallel composition of TADs.

Parallel composition allows the independent execution of the activity of the component automata except if they are intended to synchronize. We assume CSP synchronization in which actions with equal name synchronize if and only if they belong to a set of *synchronizing actions* $B \subseteq \mathcal{A}$. Since enabling of actions is determined by guards, we define the guard on the synchronized transition to be the conjunction of the guards on the synchronizing transitions. Therefore synchronization takes place only if both partners are able to execute the same synchronizing action. (Other compositions of guards are discussed in Sec. 2.6). Similarly, the deadlines of the synchronizing transitions should affect the deadline of the synchronization. In this case, we do not fix any particular operation. Instead, we assume a given operator $\otimes$ that, when applied to guards and deadlines of the synchronizing transitions, returns the deadline of the synchronization. We require that $\otimes$ satisfies the following:

1. $(\delta_1, \gamma_1) \otimes (\delta_2, \gamma_2) \Rightarrow (\gamma_1 \wedge \gamma_2)$ whenever $\delta_1 \Rightarrow \gamma_1$ and $\delta_2 \Rightarrow \gamma_2$
2. $\otimes$ preserves *left-closure*, that is, if δ_1 and δ_2 are left closed, so is $(\delta_1, \gamma_1) \otimes (\delta_2, \gamma_2)$
3. $\otimes$ distributes with respect to $\vee$ in all its arguments, that is $\left(\bigvee_i (\delta_1^i, \gamma_1^i) \otimes (\delta_2^i, \gamma_2^i) \right) \Leftrightarrow \left(\bigvee_i \delta_1^i, \bigvee_i \gamma_1^i\right) \otimes \left(\bigvee_i \delta_2^i, \bigvee_i \gamma_2^i\right)$
4. There exists a constraint $\mathbf{0}_\delta$ such that $(\mathbf{0}_\delta, \mathbf{tt})$ acts as a neutral element for $\otimes$ in the following sense: $((\delta_1, \gamma_1) \otimes (\mathbf{0}_\delta, \mathbf{tt})) \Leftrightarrow \delta_1$

$(\delta_1, \gamma_1) \otimes (\delta_2, \gamma_2)$ has to imply the guard $\gamma_1 \wedge \gamma_2$ of the resulting transition in order to preserve this property on the composed TAD. This is required in 1. Similarly, condition 2 ensures that deadlines of the composed TAD are left-closed. The distributivity of 3 is needed to prove congruence (see proof of Theorem 2.2). As we will see in the next section, time passage in a location is limited by the complement of the disjunction of the outgoing deadlines. Therefore condition 3 states compositionality for $\otimes$, allowing to represent the deadline of a synchronized action in terms of the deadlines and guards of the component automata. Constraint 4 is only necessary to show that our definition is the coarsest congruence included in the bisimulation (see Lemma 2.6). For operators not meeting these conditions there may exist coarser congruences than ours that are also bisimulation. Constraint 4 guarantees a way to test the validity of the original deadline in a component's transition by means of a synchronization. In Sec. 2.6 we discuss different implementations of $\otimes$.

Definition 2.2 *Let $T_i = (\mathcal{L}_i, \mathsf{l}^0{}_i, \mathcal{C}_i, \longrightarrow_i)$, be such that $\mathcal{C}_1 \cap \mathcal{C}_2 = \varnothing$ for $i \in \{1, 2\}$, and let $B \subseteq \mathcal{A}$ be a set of* synchronizing actions, *and $\otimes$ be an operation for synchronizing deadlines. The* parallel composition $T_1 \, ||_B^{\otimes} \, T_2$ *is defined by the TAD $(\mathcal{L}_1 \times \mathcal{L}_2, \mathsf{l}^0{}_1 \times \mathsf{l}^0{}_2, \mathcal{C}_1 \cup \mathcal{C}_2, \longrightarrow)$ where $\longrightarrow$ is defined as the smallest relation satisfying:*

$$\frac{s_1 \xrightarrow{a,\gamma,\delta,\mathbf{x}}_1 s_1' \qquad a \notin B}{(s_1, s_2) \xrightarrow{a,\gamma,\delta,\mathbf{x}} (s_1', s_2)} \qquad\qquad \frac{s_2 \xrightarrow{a,\gamma,\delta,\mathbf{x}}_2 s_2' \qquad a \notin B}{(s_1, s_2) \xrightarrow{a,\gamma,\delta,\mathbf{x}} (s_1, s_2')}$$

$$\frac{s_1 \xrightarrow{a,\gamma_1,\delta_2,\mathbf{x}_1}_1 s_1' \quad s_2 \xrightarrow{a,\gamma_2,\delta_2,\mathbf{x}_2}_2 s_2' \quad a \in B}{(s_1, s_2) \xrightarrow{a,\gamma_1\wedge\gamma_2,(\delta_1,\gamma_1)\otimes(\delta_2,\gamma_2),\mathbf{x}_1\cup\mathbf{x}_2} (s_1', s_2')}$$

The rules are fairly standard. Notice, in particular, that the last rule only allows to synchronize guards when both of them are valid. This is a significant restriction w.r.t. [BS00]. We later argue that this is nevertheless reasonable and discuss the feasibility of compositions not considered here. From now on, subscripts on edges will be omited.

Transition Systems and Bisimulation.

Definition 2.3 *A* transition system (TS for short) *is a structure* $TS = (\mathcal{S}, \mathsf{s}^0, \Sigma, \longrightarrow)$ *where*

- $\mathcal{S}$ *is an infinite set of states,*
- s^0 *is the set of initial states,*
- Σ *is a set of labels, and*
- $\longrightarrow \subseteq (\mathcal{S} \times \Sigma \times \mathcal{S})$ *is a set of transitions.*

Since we use TSs to model timed systems, we consider two kind of labels: those representing the execution of discrete actions and those representing the passage of time. Then $\Sigma = \mathcal{A} \cup \mathbb{R}_{\geq 0}$.

Definition 2.4 *A* bisimulation *[Mil89a] is a symmetric relation* $R \in \mathcal{S} \times \mathcal{S}$ *such that for all* $a \in \Sigma$*, whenever* $(p, q) \in R$ *and* $p \xrightarrow{a} p'$ *then*

$$q \xrightarrow{a} q' \text{ and } (p', q') \in R \text{ for some } q'$$

We write $p \sim q$ *if* $(p, q) \in R$ *for some bisimulation relation* R *on* TS.

Given two TSs, TS_1 and TS_2 with set of initial states, $\mathsf{s}^0{}_1$ and $\mathsf{s}^0{}_2$, respectively, we say that they are bisimilar (notation $TS_1 \sim TS_2$) if there is a bisimulation R on the disjoint union of $TS_1 \uplus TS_2$ such that $\mathsf{s}^0{}_j \subseteq R(\mathsf{s}^0{}_i)$ for $\{i, j\} = \{1, 2\}$, i.e. every initial state of TS_1 is related to some initial state of TS_2 and vice-versa.

Semantics of TADs.

In the following we recall the semantics of TADs in terms of TSs. A state of the timed system is divided in two parts, one indicating the current control location in the TAD, and the other the current time values. This last part is represented by means of a *clock valuation* which is a function $\rho : \mathcal{C} \to \mathbb{R}_{\geq 0}$ mapping to each clock the time elapsed since the last time it was reset to 0. Given a clock valuation ρ and $d \in \mathbb{R}_{\geq 0}$ the function $\rho + d$ denotes the valuation such that for each clock $x \in \mathcal{C}$, $(\rho + d)(x) = \rho(x) + d$. The function $\rho\{\mathbf{x}{:=}0\}$ denotes the valuation such that for each clock $x \in \mathbf{x} \cap \mathcal{C}$, $\rho\{\mathbf{x}{:=}0\}(x) = 0$, otherwise $\rho\{\mathbf{x}{:=}0\}(x) = \rho(x)$. We first define what it means for a constraint to be left-closed, followed by the semantics of TADs.

Definition 2.5 *A constraint ϕ is called* left closed *if and only if for all valuations ρ,*

$$\rho \models \neg\phi \;\Rightarrow\; \exists \varepsilon > 0 : \forall \varepsilon' \leq \varepsilon : \rho + \varepsilon' \models \neg\phi$$

Definition 2.6 *Let $T = (\mathcal{L}, \mathsf{l}^0, \mathcal{C}, \twoheadrightarrow)$ be a TAD. Its semantics is given by $TS(T) = (\mathcal{L} \times (\mathcal{C} \mapsto \mathbb{R}_{\geq 0}), \mathsf{l}^0 \times (\mathcal{C} \mapsto 0), \mathcal{A} \cup \mathbb{R}_{\geq 0}, \longrightarrow)$, where $\longrightarrow$ is the smallest relation satisfying:*

A1: discrete transition $s \xrightarrow{a,\gamma,\delta,\mathbf{x}} s'$ *and* $\rho \models \gamma$ *implies* $s\rho \xrightarrow{a} s'\rho\{\mathbf{x}{:=}0\}$*; and*

A2: delay transition $\forall d' < d :\ \rho + d' \models tpc(s)$ *implies* $s\rho \xrightarrow{d} s(\rho + d)$

where $tpc(s) = \neg \bigvee \{\delta \mid \exists a, \gamma, \mathbf{x}, s' : s \xrightarrow{a,\gamma,\delta,\mathbf{x}} s'\}$ *is the* time progress condition *in s.*

Rule **A1** states that an edge $s \xrightarrow{a,\gamma,\delta,\mathbf{x}} s'$ defines a discrete transition in current location s whenever the guard holds in current valuation ρ. After the transition is taken, clocks in $\mathbf{x}$ are set to 0 in the new valuation. According to **A2**, time can progress in s only when $tpc(s)$ is true, that is as long as no deadline of an edge leaving s becomes true. Notice that $tpc(s)$ is required to hold for all $d' < d$ but not for d itself. Therefore it is indistinguishable whether $tpc(s)$ holds in the limit or not. For instance, if $\rho(x) = 0$ both $x < 3$ and $x \leq 3$ hold in all $\rho + d'$ with $d' < 3$. Thus our assumption that a deadline has to be specified as a left-closed predicate is not a limitation but a preference to avoid technical complications which do not contribute to the work.

As a consequence of Def. 2.6, the notion of bisimulation extends to TADs straightforwardly: two TADs T_1 and T_2 are bisimilar (notation $T_1 \sim T_2$) if $TS(T_1) \sim TS(T_2)$.

Example 2.1 *Consider automata T_1 and T_2 of Fig. 2.1. Using Def. 2.6 it is routine to check that relation*

$$R \;=\; \{(s_0\{x{:=}d\}, t_0\{x{:=}d\}) \mid 0 \leq d\}$$

$$\cup \ \{(s_1\{x{:=}d\}, t_1\{x{:=}d\}) \mid 0 \leq d \leq 3\}$$
$$\cup \ \{(s_2\{x{:=}d\}, t_2\{x{:=}d\}) \mid 2 \leq d\}$$

is a bisimulation witnessing $T_1 \sim T_2$. Besides, if $\mathsf{stop} = (\{r\}, \{r\}, \varnothing, \varnothing)$*, then* $T_2 \mathbin{||_a^{\otimes}} \mathsf{stop}$ *can execute the trace* $b\,5\,c$*, which is not possible from* $(s_0, r)\{x{:=}0\}$*. Consequently,* $T_1 \mathbin{||_a^{\otimes}} \mathsf{stop} \not\sim T_2 \mathbin{||_a^{\otimes}} \mathsf{stop}$.

2.3 Towards a Congruence Relation

In the following we discuss different proposals for congruence until finding a satisfactory definition. All proposals are bisimulation relations on different modifications of the transition system underlying the TAD.

The example in Fig. 2.1 suggests that action c could be distinguished if time would be allowed to elapse beyond the deadline. Therefore, a first naive proposal would be to let time progress beyond the time progress condition but this would not be compatible with the bisimulation since TADs with different deadlines but equal guards may become equated. So, a modification of this semantics could consider separately a potential time progress by adding a new kind of transition as follows

$$s\rho \xrightarrow{[d]} s(\rho + d) \textit{ for all } d \geq 0.$$

Though clearly stronger than bisimulation —notice that it would distinguish T_1 and T_2 in Fig. 2.1— it fails to be a congruence. This is shown in Fig. 2.2(a). The relation would equate T_3 and T_4, but not their compositions $T_3 \mathbin{||_B^{\otimes}} T'$ and $T_4 \mathbin{||_B^{\otimes}} T'$ with $B = \{a, b, c\}$. Notice that after realization of action a, $T_3 \mathbin{||_B^{\otimes}} T'$ lets (non-potential) time progress beyond 2 time units while this is not possible in $T_4 \mathbin{||_B^{\otimes}} T'$ due to the deadline on b.

As a consequence, we may think to consider different potential time progress transitions for each edge in the TAD, but this turns out to be too strong (apart from cumbersome). See automata T_5 and T_6 in Fig. 2.2(b) which share some similitude with the previous example, only that c has been renamed to b. They are expected to be congruent.

The new example suggests that time can potentially progress differently for every action name since they can be delayed or preempted independently. A possible solution seems to consider a different kind of potential time progress for each action. Since time progress is associated to deadlines, we follow a different approach: instead of considering potential time progress, we consider a new type of discrete action ∇_D, $D \subseteq \mathcal{A}$, that indicates that from the moment action ∇_D is issued, deadlines of actions in D would be disregarded. We call this type of action "drop" (since it drops the deadline). Notice that a drop action can be performed at any moment.

Let $\mathcal{A}_\nabla = \{\nabla_D \mid D \subseteq \mathcal{A}\}$. To keep track of which deadlines have to be disregarded, states also need to book keep the current set of actions whose

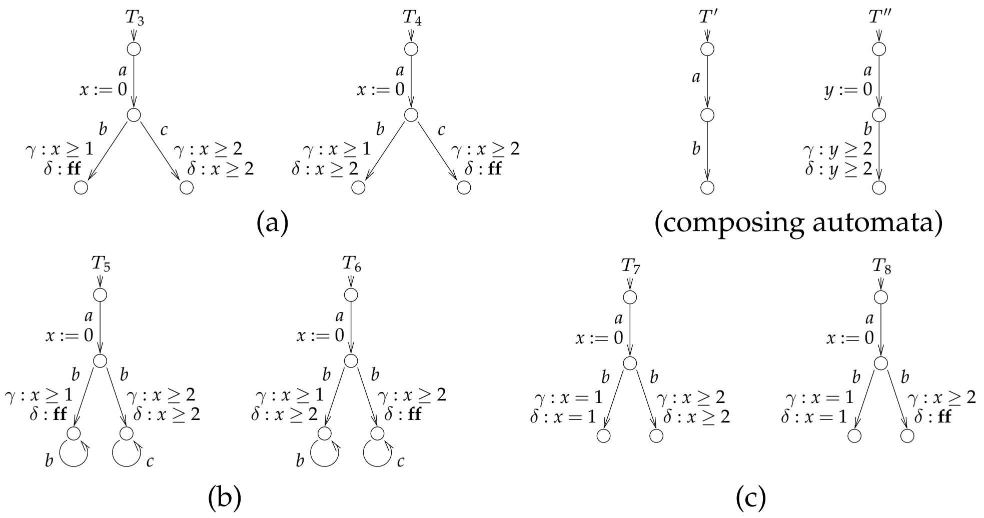

Figure 2.2: (Counter)examples for congruence

deadlines were dropped. The extended semantics of $T = (\mathcal{L}, \mathsf{l}^0, \mathcal{C}, \twoheadrightarrow)$ is then given by the TS $(\mathcal{L} \times 2^{\mathcal{A}} \times (\mathcal{C} \mapsto \mathbb{R}_{\geq 0}), \mathsf{l}^0 \times \{\varnothing\} \times (\mathcal{C} \mapsto 0), \mathcal{A} \cup \mathcal{A}_{\nabla} \cup \mathbb{R}_{\geq 0}, \longrightarrow)$, where $\longrightarrow$ is the smallest relation satisfying:

A1$_\nabla$: discrete transition

$$s \xrightarrow{a,\gamma,\delta,\mathbf{x}} s' \textit{ and } \rho \models \gamma \textit{ implies } (s, D)\rho \xrightarrow{a} (s', \varnothing)\rho\{\mathbf{x}{:=}0\}$$

A2$_\nabla$: delay transition

$$\forall d'{<}d : \rho{+}d' \models \neg dl(s, \mathcal{A} - D) \textit{ implies } (s, D)\rho \xrightarrow{d} (s, D)(\rho{+}d)$$

A3: drop transition

$$(s, D)\rho \xrightarrow{\nabla_E} (s, D \cup E)\rho$$

where $dl(s, A)$ is the deadline collected by actions in $A \subseteq \mathcal{A}$ in location s and is defined by

$$dl(s, A) = \bigvee\{\delta \mid s \xrightarrow{a,\gamma,\delta,\mathbf{x}} s' \text{ and } a \in A \text{ for some } a, \gamma, \mathbf{x}, s'\}.$$

Bisimulation in this new semantics distinguish automata in Figs. 2.1(a) and 2.2(a), and equate those in Fig. 2.2(b). Regarding the new predicate $dl(s, A)$ notice that for any location s, $tpc(s) = \neg dl(s, \mathcal{A})$.

Notice also that once a deadline is dropped, it cannot be observed anymore. The example in Fig. 2.2(c) shows that this semantics does not yet yield a congruence. According to this semantics, T_7 and T_8 are equated.

However, under the assumption that deadlines of synchronising transitions are arranged in a conjunction[1] (i.e. $\otimes$ is $\wedge$), the compositions $T_7 \,||_B^{\otimes}\, T''$ and $T_8 \,||_B^{\otimes}\, T''$, with $B = \{a, b\}$, are distinguished by the usual bisimulation: after executing action a, $T_8 \,||_B^{\otimes}\, T''$ let time progress beyond 2 time units while this is not the case in $T_7 \,||_B^{\otimes}\, T''$ due to the composed deadline $(x \geq 2) \wedge (y \geq 2)$ in b.

This phenomenon is due to the fact that after action a is performed, automaton T'' *temporarily* disregards the deadline of b during the first 2 units of time, but later it allows it to beobserved again. As a consequence, we introduce a new action Δ (read "undrop") which indicates that in the future all deadlines will be considered again.

Definition 2.7 *The extended semantics of* $T = (\mathcal{L}, \mathsf{l}^0, \mathcal{C}, \rightarrowtail)$ *is given by*

$$TS_{\nabla}(T) = (\mathcal{L} \times 2^{\mathcal{A}} \times (\mathcal{C} \mapsto \mathbb{R}_{\geq 0}), \mathsf{l}^0 \times \varnothing \times (\mathcal{C} \mapsto 0), \mathcal{A} \cup \mathcal{A}_{\nabla} \cup \{\Delta\} \cup \mathbb{R}_{\geq 0}, \longrightarrow),$$

where $\longrightarrow$ *is the smallest relation satisfying* $\mathbf{A1}_{\nabla}$, $\mathbf{A2}_{\nabla}$, *and* **A3** *above plus*

A4: undrop transition $\quad (s, D)\rho \xrightarrow{\Delta} (s', \varnothing)\rho$

Note that the undrop action can be performed at any moment. Going back to the privous example, the execution sequence $a\,\nabla_{\{b\}}\,2\,\Delta\,1$ is possible in T_8 but not in T_7. Hence, a bisimulation in this setting distinguishes T_7 from T_8. We define such a relation as follows.

Definition 2.8 (∇-bisimulation) *We say that automata* T_1 *and* T_2 *are* ∇-bisimilar, *notation* $T_1 \sim^{\nabla} T_2$, *if* $TS_{\nabla}(T_1) \sim TS_{\nabla}(T_2)$. *We also say that locations* s *and* t *are* ∇-bisimilar *in some valuation* ρ, *notation* $s\rho \sim^{\nabla} t\rho$, *if* $(s, \varnothing)\rho \sim (t, \varnothing)\rho$.

The following Propositon states that two ∇-bisimilar automata are also bisimilar.

Proposition 2.1 *For any* T_1 *and* T_2, *if* $T_1 \sim^{\nabla} T_2$ *then* $T_1 \sim T_2$.

Proof: It is routine to check that if R is a bisimulation that witness $T_1 \sim^{\nabla} T_2$, then $\{(s_1\rho_1, s_2\rho_2) \mid ((s_1, \varnothing)\rho_1, (s_2, \varnothing)\rho_2) \in R\}$ is a bisimulation that witness $T_1 \sim T_2$. $\square$

We conclude this section by stating two basic properties (lemmas) of ∇-bisimulation. They are needed to prove Theorem 2.1 which relates $\sim^{\nabla}$ to a symbolic bisimulation.

[1]Conjugating deadlines on synchronisation is an operation that can be used to model patient synchronisation (also known as flexible synchronisation in [SY96]) in soft-real time applications. For further discussion see Section 2.6 on page 42.

Notice that the ability of dropping all the deadlines, letting time pass, and then undropping the deadlines, ensures that if two locations are ∇-bisimilar at a certain moment, no matter how long the activity is blocked, these two locations will still be ∇-bisimilar. This is stated in Lemma 2.1. Moreover, if two locations are ∇-bisimilar at some given valuation ρ then both satisfy the deadline associated to some action in valuation ρ, or neither of them does. This is easy to check by dropping all the deadlines except those associated to the action of interest. This is formally stated in Lemma 2.2.

Lemma 2.1 *If $t\rho \sim^{\nabla} u\rho$ then $t(\rho + d) \sim^{\nabla} u(\rho + d)$, for all $d \geq 0$.*

Proof: By **A3** in Def. 2.7, $(t, \varnothing)\rho \xrightarrow{\nabla_{\mathcal{A}}} (t, \mathcal{A})\rho$ which implies $(u, \varnothing)\rho \xrightarrow{\nabla_{\mathcal{A}}} (u, \mathcal{A})\rho$ and $(t, \mathcal{A})\rho \sim (u, \mathcal{A})\rho$ ($\sim$ is a bisimulation.)

Since $dl(t, \varnothing) = \bigvee \varnothing = \mathbf{ff}$, $\rho + d \models \neg dl(t, \mathcal{A} - \mathcal{A})$, for all $d \geq 0$. Hence, by **A2**$_{\nabla}$, $(t, \mathcal{A})\rho \xrightarrow{d} (t, \mathcal{A})(\rho + d)$ for any $d \geq 0$. This implies that $(u, \mathcal{A})\rho \xrightarrow{d} (u, \mathcal{A})(\rho + d)$ and $(t, \mathcal{A})(\rho + d) \sim (u, \mathcal{A})(\rho + d)$ for all $d \geq 0$.

Finally, since $(t, \mathcal{A})(\rho + d) \xrightarrow{\Delta} (t, \varnothing)(\rho + d)$ by **A4**, $(u, \mathcal{A})(\rho + d) \xrightarrow{\Delta} (u, \varnothing)(\rho + d)$ and $(t, \varnothing)(\rho + d) \sim (u, \varnothing)(\rho + d)$ for all $d \geq 0$. □

Lemma 2.2 *If $t\rho \sim^{\nabla} u\rho$ then $\rho \models dl(t, D) \Leftrightarrow dl(u, D)$, for any $D \subseteq \mathcal{A}$.*

Proof: Let $(t, \varnothing)\rho \sim (u, \varnothing)\rho$. By **A3** $(t, \varnothing)\rho \xrightarrow{\nabla_{\mathcal{A}-D}} (t, \mathcal{A} - D)\rho$, which implies that $(u, \varnothing)\rho \xrightarrow{\nabla_{\mathcal{A}-D}} (u, \mathcal{A} - D)\rho$ and $(t, \mathcal{A} - D)\rho \sim (u, \mathcal{A} - D)\rho$.

We show that $(t, \mathcal{A} - D)\rho \sim (u, \mathcal{A} - D)\rho$ implies $\rho \models dl(t, D) \Rightarrow dl(u, D)$ which, by symmetry of $\sim$, suffices to show that $\rho \models dl(t, D) \Leftrightarrow dl(u, D)$.

$$\begin{array}{lr}
\rho \models \neg dl(u, D) & \\
\Rightarrow\ \exists d > 0 :\ \forall d' :\ 0 \leq d' < d\ :\ \rho + d' \models \neg dl(u, D) & \text{(by Def. 2.5)} \\
\Rightarrow\ (u, \mathcal{A} - D)\rho \xrightarrow{d} & \text{(by } \mathbf{A2}_{\nabla}\text{)} \\
\Rightarrow\ (t, \mathcal{A} - D)\rho \xrightarrow{d} & \text{(since } (t, \mathcal{A} - D)\rho \sim (u, \mathcal{A} - D)\rho\text{)} \\
\Rightarrow\ \exists d > 0 :\ \forall d' :\ 0 \leq d' < d\ :\ \rho + d' \models \neg dl(t, D) & \text{(by } \mathbf{A2}_{\nabla}\text{)} \\
\rho \models \neg dl(t, D) & \text{(in particular, for } d' = 0\text{)}
\end{array}$$

□

2.4 Symbolic Characterization of ∇-bisimulation

We postpone the proof that ∇-bisimulation is a congruence until Sec. 2.5 and give first a symbolic characterization of $\sim^{\nabla}$. That is, we give a relation directly on TADs which does not resort to the underlying transition system and equates exactly the same automata as $\sim^{\nabla}$ does. The symbolic bisimulation we propose works in a similar fashion to that of [LY02]. The construction of such a relation is based on zone and region manipulation. A clock region or *region* for short, is a consistent conjunction of atomic constraints of the form,

$$\psi \equiv \bigwedge_{x \in C} \psi_x \wedge \bigwedge_{\{x,y\} \subseteq C, x \neq y} \psi_{\{x,y\}}$$

where

- each ψ_x is either $x = n$, $m < x < m+1$ or $x > N$, and
- each $\psi_{\{x,y\}}$ is either $x - y = n$, $m < x - y < m+1$ or $x - y > N$.

with n, m, N non-negative integers such that $0 \leq n \leq N$, and $0 \leq m < N$. Regions can be expressed by constraints as we defined above, and any constraint can be expressed as a disjunction of regions. Similar to the clock resetting ($\rho\{\mathbf{x} := 0\}$) and time successor ($\rho + d$) of the clock valuation defined earlier, we define below their symbolic counterpart.

Reset: For a constraint ϕ and a set of clocks $\mathbf{x}$, the reset $\phi{\downarrow_{\mathbf{x}}}$ is a predicate such that for all ρ, $\rho \models \phi{\downarrow_{\mathbf{x}}}$ iff $\rho = \rho'\{\mathbf{x} := 0\}$ and $\rho' \models \phi$ for some ρ'.

Time successor: For a constraint ϕ, the time successor $\phi\Uparrow$ is a predicate such that for all ρ, $\rho \models \phi\Uparrow$ iff $\rho = \rho' + d$ and $\rho' \models \phi$ for some ρ' and $d \geq 0$.

A constraint ϕ *is* $\Uparrow$*-closed* if and only if $\phi\Uparrow \Leftrightarrow \phi$ is valid (i.e. a tautology). The operations above distribute through disjunction and are expressible in terms of constraints (see e.g. [Yov98, LY02].) The following facts can be derived from the definitions or have already appeared elsewhere [Yov98, LY02].

Fact 2.1

1. *Let ψ and ϕ be regions. Let ρ and ρ' be valuations s.t. $\rho \models \psi$ and $\rho' \models \psi$. If $\rho + d \models \phi$ for some $d \geq 0$, then there exists $d' \geq 0$ s.t. $\rho' + d' \models \phi$.*
2. *If ϕ is a region then, for any constraint ψ, either $\phi \Rightarrow \psi$ is valid or $\phi \wedge \psi$ is a contradiction.*
3. *If ϕ is a region, so is $\phi{\downarrow_{\mathbf{x}}}$.*
4. *$\rho \models \phi$ implies $\rho \models \phi\Uparrow$.*
5. *$\phi\Uparrow$ is $\Uparrow$-closed.*
6. *If ϕ is $\Uparrow$-closed then $\rho \models \phi$ implies $\rho + d \models \phi$ for all $d \in \mathbb{R}_{\geq 0}$.*
7. *If ϕ_1 and ϕ_2 are $\Uparrow$-closed (resp. left-closed), so are $\phi_1 \wedge \phi_2$ and $\phi_1 \vee \phi_2$.*

Given a constraint ϕ, a ϕ-partition [LY02] is a finite set of constraints Φ if $\bigvee \Phi \Leftrightarrow \phi$ and for any two distinct $\psi, \psi' \in \Phi$, ψ and ψ' are disjoint (i.e. $\psi \wedge \psi'$ is a contradiction). A ϕ-partition Φ is called *finer* than another ϕ-partition Ψ if Φ can be obtained from Ψ by decomposing some of its elements. $\mathcal{RC}(\phi)$ denotes the set of all regions that constitute ϕ. Notice that $\phi \Leftrightarrow \bigvee \mathcal{RC}(\phi)$ and that $\mathcal{RC}(\phi)$ is the finest of all ϕ-partitions.

Lemma 2.3 *Let ψ be a region and ρ be such that $\rho \models \psi$. For all $\phi \in \mathcal{RC}(\psi\Uparrow)$ exists $d \geq 0$ such that $\rho + d \models \phi$.*

Proof: Let $\rho'' \models \phi$, then $\rho'' \models \psi\Uparrow$. By the definition of$\Uparrow$, exists ρ' and $d' \geq 0$ such that $\rho' + d' = \rho''$ and $\rho' \models \psi$. Since $\rho \models \psi$ too, and ψ and ϕ are regions, by Fact 2.1.1, exists $d \geq 0$ such that $\rho + d \models \phi$. □

The definition of symbolic bisimulation we propose is based on Lin & Yi's definition [LY02], which in turns is based on Čerāns' result [Čerāns92]. A symbolic bisimulation is a relation containing tuples (s, t, ϕ) meaning that locations s and t are related in any valuation that satisfies constraint ϕ. Here ϕ is a constraint over the disjoint union of the set of clocks of the two automata. In this way, the relation ensures that clocks in both automata progress at the same rate. In turn, this guarantees that the related locations can idle the same time until some given deadline becomes true.

Definition 2.9 (Symbolic Bisimulation) *Let T_1 and T_2 be two TADs with disjoint set of clocks $\mathcal{C}_1$ and $\mathcal{C}_2$ and disjoint set of locations $\mathcal{L}_1$ and $\mathcal{L}_2$ respectively. A relation $S \subseteq (\mathcal{L}_1 \times \mathcal{L}_2 \cup \mathcal{L}_2 \times \mathcal{L}_1) \times \mathcal{F}(\mathcal{C}_1 \cup \mathcal{C}_2)$ (where $\mathcal{F}(\mathcal{C})$ denotes the set of all constraints with clocks in $\mathcal{C}$) is a* symbolic bisimulation *if for all $(t, u, \phi) \in S$,*

1. *$(u, t, \phi) \in S$,*
2. *ϕ is $\Uparrow$-closed,*
3. *whenever $t \xrightarrow{a,\gamma,\delta,\mathbf{x}} t'$, there is a $(\phi \wedge \gamma)$-partition Φ such that for each $\phi' \in \Phi$, $u \xrightarrow{a,\gamma',\delta',\mathbf{y}} u'$, $\phi' \Rightarrow \gamma'$ and $(t', u', \phi'{\downarrow_{\mathbf{xy}}}\Uparrow) \in S$, for some γ', δ', $\mathbf{y}$ and u'; and*
4. *$\phi \Rightarrow (dl(t, A) \Leftrightarrow dl(u, A))$ is valid for all $A \subseteq \mathcal{A}$.*

We write $t \sim^{\phi} u$ if $(t, u, \phi) \in S$ for some symbolic bisimulation S. We also write $T_1 \sim^{\phi} T_2$ if for every initial location t of T_1 there is an initial location u in T_2 such that $t \sim^{\phi} u$, and the same with the roles of T_1 and T_2 exchanged.

Property 1 states the symmetric characteristics of a bisimulation. The requirement that ϕ is $\Uparrow$-closed (property 2) ensures that location t and u show an equivalent behavior any time in the future which is necessary if deadlines are dropped. Property 3 ensures the transfer properties of discrete transitions. This is similar to [LY02] except that there is no invariant to

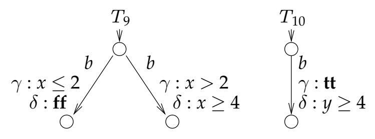

Figure 2.3: $T_9 \sim^{x=y} T_{10}$

consider. Finally, property 4 states that any possible combination of deadlines should match under the assumption that ϕ holds. This ensures that the time elapsed until a deadline associated to a given action is the same in both locations. Notice that property 4 is equivalent to requiring that $\phi \Rightarrow (dl(t, \{a\}) \Leftrightarrow dl(u, \{a\}))$ for all $a \in \mathcal{A}$. This makes evident that deadlines may be "changed" from one edge to another as long as both edges are labeled with the same action (see Fig. 2.2(b)). Moreover property 4 is comparable to the property of invariants in [LY02]. Like in [LY02], the use of partitioning allows that one edge is matched by several edges as is the case in Fig. 2.3 where both $T_9 \sim^{\nabla} T_{10}$ and $T_9 \sim^{x=y} T_{10}$.

The following theorem states that symbolic bisimulation completely captures the notion of ∇-bisimulation.

Theorem 2.1 *For $\Uparrow$-closed ϕ, $t \sim^{\phi} u$ iff $t\rho \sim^{\nabla} u\rho$ for any $\rho \models \phi$*

Proof: ($\Rightarrow$) Let S be a symbolic bisimulation. Define

$$R \;=\; \{((t,D)\rho,(u,D)\rho) \mid \exists\phi : \rho \models \phi : (t,u,\phi) \in S \text{ and } D \subseteq \mathcal{A}\} \qquad (2.1)$$

We show that R is bisimulation. The fact that it is symmetric follows by symmetry of S. In the following we suppose that $((t,D)\rho,(u,D)\rho) \in R$ as a consequence of $(t,u,\phi) \in S$ as indicated in (2.1), and prove the transfer property by doing case analysis on the type of transition.

discrete transition:

$$
\begin{aligned}
&(t,D)\rho \xrightarrow{a} (t',D')\rho' \\
&\Rightarrow\ \{\text{by } \mathbf{A1}_{\nabla}\} \\
&\quad \exists\gamma,\delta,\mathbf{x} :\ t \xrightarrow{a,\gamma,\delta,\mathbf{x}} t',\ D' = \varnothing,\ \rho' = \rho\{\mathbf{x} := 0\},\ \text{ and }\ \rho \models \gamma \qquad (2.2)\\
&\Rightarrow\ \{\text{by prop. 3 in Def. 2.9, since } (t,u,\phi) \in S\} \\
&\quad \exists\Phi : \Phi \text{ is a } (\phi \wedge \gamma)\text{-partition} : \forall\phi' \in \Phi : \\
&\qquad\qquad u \xrightarrow{a,\gamma',\delta',\mathbf{x}'} u',\ \phi' \Rightarrow \gamma',\ (t',u',\phi'{\downarrow_{\mathbf{x}\mathbf{x}'}}\Uparrow) \in S,
\end{aligned}
$$

[By (2.1) $\rho \models \phi$ and by (2.2) $\rho \models \gamma$, hence $\rho \models \phi \wedge \gamma$. Since Φ is a $(\phi \wedge \gamma)$-partition, then $\rho \models \phi'$ for some $\phi' \in \Phi$. Finally, since $\phi' \Rightarrow \gamma'$ the $\rho \models \gamma'$ also holds.]

$\Rightarrow$ {by observation}

$$\begin{array}{l}
\quad u \xrightarrow{a,\gamma',\delta',\mathbf{x}'} u',\ \rho \models \gamma',\ \rho \models \phi',\ \text{ and }\ (t', u', \phi'{\downarrow_{\mathbf{xx}'}}\Uparrow) \in S \\
\Rightarrow\ \{\text{by } \mathbf{A1}_\nabla\text{, def. of } \downarrow_{\mathbf{xx}'}\text{, and Fact 2.1.4}\} \\
\quad (u, D)\rho \xrightarrow{a} (u', \varnothing)\rho\{\mathbf{xx}' := 0\}, \\
\quad \rho\{\mathbf{x}' := 0\} \models \phi'{\downarrow_{\mathbf{xx}'}}\Uparrow,\ \text{ and }\ (t', u', \phi'{\downarrow_{\mathbf{xx}'}}\Uparrow) \in S \\
\Rightarrow\ \{\text{by (2.1)}\} \\
\quad (u, D)\rho \xrightarrow{a} (u', \varnothing)\rho\{\mathbf{x}' := 0\}\ \text{ and} \\
\quad ((t', \varnothing)\rho\{\mathbf{xx}' := 0\}, (u', \varnothing)\rho\{\mathbf{xx}' := 0\}) \in R \\
\Rightarrow\ \{\text{by def. of reset}\} \\
\quad (u, D)\rho \xrightarrow{a} (u', \varnothing)\rho\{\mathbf{x}' := 0\}\ \text{ and,} \\
\quad\quad ((t', \varnothing)\rho\{\mathbf{x} := 0\}, (u', \varnothing)\rho\{\mathbf{x}' := 0\}) \in R
\end{array}$$

delay transition:

$$\begin{array}{l}
(t, D)\rho \xrightarrow{d} (t', D')\rho' \\
\Rightarrow\ \{\text{by } \mathbf{A2}_\nabla\} \\
\quad \forall d' < d :\ \rho + d' \models \neg dl(t, \mathcal{A} - D),\ t = t',\ D' = D,\ \text{ and }\ \rho' = \rho + d
\end{array}$$

$$\left[\ \begin{array}{l}
\text{By (2.1), } \rho \models \phi \text{ for some } \phi \text{ s.t. } (t, u, \phi) \in S\text{. Moreover, by Fact 2.1.6,} \\
\rho + d' \models \phi \text{ for all } d' \geq 0\text{, in particular if } d' < d\text{. As a consequence of} \\
\text{prop. 4 in Def. 2.9, } \rho + d' \models dl(t, \mathcal{A} - D) \Leftrightarrow dl(u, \mathcal{A} - D).
\end{array}\ \right]$$

$$\begin{array}{l}
\Rightarrow\ \{\text{by observation}\} \\
\quad \forall d' < d :\ \rho + d' \models \neg dl(u, \mathcal{A} - D) \\
\Rightarrow\ \{\text{by } \mathbf{A2}_\nabla\} \\
\quad (u, D)\rho \xrightarrow{d} (u, D)(\rho + d) \\
\Rightarrow\ \left\{\begin{array}{l}\text{by (2.1), since } (t, u, \phi) \in S \text{ and } \rho + d \models \phi \text{ (see previous} \\ \text{observation)}\end{array}\right\} \\
\quad (u, D)\rho \xrightarrow{d} (u, D)(\rho + d)\ \text{ and }\ ((t, D)(\rho + d), (u, D)(\rho + d)) \in R
\end{array}$$

drop transition: Notice that both $(t, D)\rho \xrightarrow{\nabla_E} (t, D \cup E)\rho$ and $(u, D)\rho \xrightarrow{\nabla_E} (u, D \cup E)\rho$, by **A3**. Moreover, since $(t, u, \phi) \in S$ and $\rho \models \phi$, by (2.1), $((t, D \cup E)\rho, (u, D \cup E)\rho) \in R$.

undrop transition: Similarly, $(t, D)\rho \xrightarrow{\Delta} (t, \varnothing)\rho$ and $(u, D)\rho \xrightarrow{\Delta} (u, \varnothing)\rho$, by **A4**. Moreover, since $(t, u, \phi) \in S$ and $\rho \models \phi$, by (2.1), $((t, \varnothing)\rho, (u, \varnothing)\rho) \in R$.

($\Leftarrow$) We prove that relation

$$S = \{\ (t, u, \phi) \mid \begin{array}{l} \phi \text{ is } \Uparrow\text{-closed and} \\ \forall \psi \in \mathcal{RC}(\phi) :\ \exists \rho :\ \rho \models \psi\ :\ (t, \varnothing)\rho \sim (u, \varnothing)\rho\ \} \end{array} \tag{2.3}$$

is a symbolic bisimulation. Since $\sim$ is symmetric, S satisfies prop. 1 of Def. 2.9 and by definition, it satisfies prop. 2 as well. In the following we prove that S also satisfies properties 3 and 4 in Def. 2.9.

Property 3:

$$t \xrightarrow{a,\gamma,\delta,\mathbf{x}} t' \text{ and } (t,u,\phi) \in S$$
$$\Rightarrow \{\text{by (2.3)}\}$$
$$t \xrightarrow{a,\gamma,\delta,\mathbf{x}} t' \text{ and } \forall\psi \in \mathcal{RC}(\phi) : \exists\rho : \rho \models \psi : (t,\varnothing)\rho \sim (u,\varnothing)\rho$$

$$\left[\begin{array}{l}\text{Take } \Phi = \mathcal{RC}(\phi \wedge \gamma). \text{ Notice that it is a } (\phi \wedge \gamma)\text{-partition and that}\\ \Phi \subseteq \mathcal{RC}(\phi) \text{ by Fact 2.1.2. Then } \psi \Rightarrow \gamma \text{ for all } \psi \in \Phi.\end{array}\right]$$

$$\Rightarrow \{\text{by observation}\}$$
$$t \xrightarrow{a,\gamma,\delta,\mathbf{x}} t' \text{ and } \forall\psi \in \Phi : \exists\rho : \rho \models \psi : (t,\varnothing)\rho \sim (u,\varnothing)\rho \wedge \rho \models \gamma$$
$$\Rightarrow \{\text{by } \mathbf{A1}_\nabla\}$$
$$\forall\psi \in \Phi : \exists\rho : \rho \models \psi :$$
$$(t,\varnothing)\rho \sim (u,\varnothing)\rho \text{ and } (t,\varnothing)\rho \xrightarrow{a} (t',\varnothing)\rho\{\mathbf{x} := 0\}$$
$$\Rightarrow \{\sim \text{ is a bisimulation}\}$$
$$\forall\psi \in \Phi : \exists\rho : \rho \models \psi :$$
$$(u,\varnothing)\rho \xrightarrow{a} (u',D)\rho' \text{ and } (t',\varnothing)\rho\{\mathbf{x} := 0\} \sim (u',D)\rho'$$
$$\Rightarrow \{\text{by } \mathbf{A1}_\nabla\}$$
$$\forall\psi \in \Phi : \exists\rho : \rho \models \psi :$$
$$u \xrightarrow{a,\gamma',\delta',\mathbf{x}'} u', \ \rho \models \gamma', \ D' = \varnothing, \ \rho' = \rho\{\mathbf{x}' := 0\},$$
$$\text{and } (t',\varnothing)\rho\{\mathbf{x} := 0\} \sim (u',\varnothing)\rho\{\mathbf{x}' := 0\}$$

$$\left[\begin{array}{l}\text{Since } \rho \models \psi \text{ and } \rho \models \gamma', \psi \wedge \gamma' \text{ is not a contradiction and hence } \psi \Rightarrow \gamma'\\ \text{by Fact 2.1.2.}\end{array}\right]$$

$$\Rightarrow \{\text{by observation}\}$$
$$\forall\psi \in \Phi : \exists\rho : \rho \models \psi : u \xrightarrow{a,\gamma',\delta',\mathbf{x}'} u', \ \psi \Rightarrow \gamma',$$
$$\text{and } (t',\varnothing)\rho\{\mathbf{x} := 0\} \sim (u',\varnothing)\rho\{\mathbf{x}' := 0\}$$
$$\Rightarrow \{\text{by def. of reset}\}$$
$$\forall\psi \in \Phi : u \xrightarrow{a,\gamma',\delta',\mathbf{x}'} u', \ \psi \Rightarrow \gamma', \text{ and}$$
$$\exists\rho : \rho \models \psi : (t',\varnothing)\rho\{\mathbf{x}\mathbf{x}' := 0\} \sim (u',\varnothing)\rho\{\mathbf{x}\mathbf{x}' := 0\}$$
$$\Rightarrow \{\text{by Def. of } \downarrow_{\mathbf{x}\mathbf{x}'}\}$$
$$\forall\psi \in \Phi : u \xrightarrow{a,\gamma',\delta',\mathbf{x}'} u', \ \psi \Rightarrow \gamma', \text{ and}$$

$$\exists\rho:\ \rho\{\mathbf{xx}' := 0\} \models \psi{\downarrow_{\mathbf{xx}'}} :\ (t',\varnothing)\rho\{\mathbf{xx}' := 0\} \sim (u',\varnothing)\rho\{\mathbf{xx}' := 0\}$$

$\Rightarrow$ {by Lemma 2.3, since $\psi{\downarrow_{\mathbf{xx}'}}$ is a region by Fact 2.1.3}

$$\forall\psi \in \Phi:\ u \xrightarrow{a,\gamma',\delta',\mathbf{x}'} u',\ \ \psi \Rightarrow \gamma',\ \text{ and}$$
$$\exists\rho:\ \forall\xi \in \mathcal{RC}(\psi{\downarrow_{\mathbf{xx}'}}{\Uparrow}):\ \exists d \geq 0:\ (\rho\{\mathbf{xx}' := 0\} + d) \models \xi$$
$$\text{and } (t',\varnothing)\rho\{\mathbf{xx}' := 0\} \sim (u',\varnothing)\rho\{\mathbf{xx}' := 0\}$$

$\Rightarrow$ {by Lemma 2.1 and logics}

$$\forall\psi \in \Phi:\ u \xrightarrow{a,\gamma',\delta',\mathbf{x}'} u',\ \ \psi \Rightarrow \gamma',\ \text{ and}$$
$$\forall\xi \in \mathcal{RC}(\psi{\downarrow_{\mathbf{xx}'}}{\Uparrow}):\ \exists\rho:\ \exists d \geq 0:$$
$$(\rho\{\mathbf{xx}' := 0\} + d) \models \xi \text{ and}$$
$$(t',\varnothing)(\rho\{\mathbf{xx}' := 0\} + d) \sim (u',\varnothing)(\rho\{\mathbf{xx}' := 0\} + d)$$

$\Rightarrow$ {taking $\rho' = \rho\{\mathbf{xx}' := 0\} + d$}

$$\forall\psi \in \Phi:\ u \xrightarrow{a,\gamma',\delta',\mathbf{x}'} u',\ \ \psi \Rightarrow \gamma',\ \text{ and}$$
$$\forall\xi \in \mathcal{RC}(\psi{\downarrow_{\mathbf{xx}'}}{\Uparrow}):\ \exists\rho':\ \rho' \models \xi \text{ and } (t',\varnothing)\rho' \sim (u',\varnothing)\rho'$$

$\Rightarrow$ {by (2.3), since $\psi{\downarrow_{\mathbf{xx}'}}{\Uparrow}$ is ${\Uparrow}$-closed}

$$\forall\psi \in \Phi:\ u \xrightarrow{a,\gamma',\delta',\mathbf{x}'} u',\ \ \psi \Rightarrow \gamma',\ \text{ and } (t',u',\psi{\downarrow_{\mathbf{xx}'}}{\Uparrow}) \in S$$

Property 4:

$$(t,u,\phi) \in S$$

$\Rightarrow$ {by (2.3)}

$$\forall\psi \in \mathcal{RC}(\phi):\ \exists\rho:\ \rho \models \psi\ :\ (t,\varnothing)\rho \sim (u,\varnothing)\rho$$

$\Rightarrow$ {by Lemma 2.2}

$$\forall\psi \in \mathcal{RC}(\phi):\ \exists\rho:\ \rho \models \psi\ :\ \forall D \subseteq \mathcal{A}:\ \rho \models dl(t,D) \Leftrightarrow dl(u,D)$$

[$\rho \models \psi$ and $\rho \models dl(t,D) \Leftrightarrow dl(u,D)$ implies $\psi \wedge (dl(t,D) \Leftrightarrow dl(u,D))$ is not a contradiction.]

$\Rightarrow$ {by Fact 2.1.2 and previous observation}

$$\forall\psi \in \mathcal{RC}(\phi):\ \forall D \subseteq \mathcal{A}:\ \psi \Rightarrow (dl(t,D) \Leftrightarrow dl(u,D))$$

$\Rightarrow$ {by logics using the fact that $\phi \Leftrightarrow \bigvee \mathcal{RC}(\phi)$}

$$\forall D \subseteq \mathcal{A}:\ \phi \Rightarrow (dl(t,D) \Leftrightarrow dl(u,D))$$

□

Corollary 2.1 *Let* $\phi_0 \equiv \bigwedge_{x,y \in \mathcal{C}_1 \cup \mathcal{C}_2}(0 \leq x = y)$. $T_1 \sim^{\phi_0} T_2$ *iff* $T_1 \sim^{\nabla} T_2$.

2.5 The Coarsest Congruence Included in $\sim$

In this section, we shall show that $\sim^{\phi_0}$ (and hence $\sim^{\nabla}$, too) is the coarsest congruence for the parallel composition included in bisimulation. The first part of the section is devoted to proving that $\sim^{\phi_0}$ is a congruence. It is interesting to notice that the proof of congruence is carried out fully at the symbolic level (in contrast to the usual proof using the underlying transition system). To the best of our knowledge, this is a novel approach. In the second part we show that $\sim^{\nabla}$ is the coarsest congruence included in $\sim$.

The next two lemmas are required for the proof of congruence. Lemma 2.4 implies that a deadline of a set of actions can be decomposed as a disjunction of the deadlines of each of the actions. Lemma 2.5 states that if two locations t and u are symbolically bisimilar under a constraint ϕ, then a given action a is enabled in t if and only if it is enabled in u for all valuations that satisfy constraint ϕ. In particular, these lemmas are needed to check that property 4 of the symbolic bisimulation is preserved in the congruence.

Lemma 2.4 $dl(s, D \cup E) \Leftrightarrow (dl(s, D) \vee dl(s, E))$

Lemma 2.5 *Define* $gd(s, a) = \bigvee\{\gamma \mid s \xrightarrow{a,\gamma,\delta,\mathbf{x}} s'$ *for some* $\delta, \mathbf{x}, s'\}$. *If S is a symbolic bisimulation s.t.* $(t, u, \phi) \in S$, *then* $\phi \Rightarrow (gd(t, a) \Leftrightarrow gd(u, a))$ *is valid for all* $a \in \mathcal{A}$.

Proof: Let S be a symbolic bisimulation with $(t, u, \phi) \in S$. By symmetry (property 1, Def. 2.9), it suffices to show that $\phi \Rightarrow (gd(t, a) \Rightarrow gd(u, a))$. By definition of gd, this follows by the claim that, for all γ such that $t \xrightarrow{a,\gamma,\delta,\mathbf{x}} t'$, $\phi \Rightarrow (\gamma \Rightarrow gd(u, a))$ (that is $(\phi \wedge \gamma) \Rightarrow gd(u, a)$) which is what we prove in the following.

$$
\begin{array}{l}
t \xrightarrow{a,\gamma,\delta,\mathbf{x}} t' \\
\Rightarrow \ \{\text{by prop. 3 in Def. 2.9, since } (t, u, \phi) \in S\} \\
\quad \exists \Phi : \Phi \text{ is a } (\phi \wedge \gamma)\text{-partition} : \forall \phi' \in \Phi : \ u \xrightarrow{a,\gamma',\delta',\mathbf{x}'} u' \ \text{ and } \ \phi' \Rightarrow \gamma' \\
\Rightarrow \ \{\gamma' \Rightarrow gd(u, a) \text{ by def. of } gd\} \\
\quad \exists \Phi : \Phi \text{ is a } (\phi \wedge \gamma)\text{-partition} : \forall \phi' \in \Phi : \ \phi' \Rightarrow gd(u, a) \\
\Rightarrow \ \{(\phi \wedge \gamma) \Leftrightarrow \bigvee \Phi\} \\
\quad (\phi \wedge \gamma) \Rightarrow gd(u, a)
\end{array}
$$

□

Now, we are in conditions to prove that $\sim^{\phi}$ is a congruence for any parallel composition defined as in Sec. 2.2. In particular, we notice that the proof does not use constraints 1 and 4 imposed on $\otimes$.

Theorem 2.2 *Let $T_i^j = (\mathcal{L}_i^j, \mathsf{l}0_i^j, \mathcal{C}_i^j, \twoheadrightarrow)$, for $i, j \in \{1,2\}$ such that $\mathcal{C}_i^j \cap \mathcal{C}_k^l = \varnothing$ if $i \neq k$ or $j \neq l$. Then $T_1^1 \sim^\phi T_2^1$ and $T_1^2 \sim^\phi T_2^2$ imply $T_1^1 \,||_B^\otimes\, T_1^2 \sim^\phi T_2^1 \,||_B^\otimes\, T_2^2$ for all $B \in \mathcal{A}$, operation $\otimes$ and constraint ϕ.*

Proof: Let S_1 and S_2 be two symbolic bisimulations that witness $T_1^1 \sim^\phi T_2^1$ and $T_1^2 \sim^\phi T_2^2$ respectively. Define

$$S = \{((t_1,t_2),(u_1,u_2),\phi_1 \wedge \phi_2) \mid (t_1,u_1,\phi_1) \in S_1 \text{ and } (t_2,u_2,\phi_2) \in S_2\} \qquad (2.4)$$

We prove that S is also a symbolic bisimulation from which the theorem follows. For this, we check that $((t_1,t_2),(u_1,u_2),\phi_1 \wedge \phi_2) \in S$, obtained as in (2.4), satisfy the four properties in Def. 2.9. Property 1 is immediate since S_1 and S_2 also satisfy it. So is property 2: since ϕ_1 and ϕ_2 are $\Uparrow$-closed, so is $\phi_1 \wedge \phi_2$ using Fact 2.1.7. We proceed to check the remaining two properties.

Property 3: Suppose $(t_1,t_2) \xrightarrow{a,\gamma,\delta,\mathbf{x}} (t_1',t_2')$. Then three cases arise

Case: $a \notin B$ with $t_1 \xrightarrow{a,\gamma,\delta,\mathbf{x}} t_1'$ and $t_2' = t_2$]

$t_1 \xrightarrow{a,\gamma,\delta,\mathbf{x}} t_1'$

$\Rightarrow$ {by prop. 3 in Def. 2.9, since $(t_1,u_1,\phi_1) \in S_1$}

$\exists\Phi : \Phi$ is a $(\phi_1 \wedge \gamma)$-partition :

$\forall\phi \in \Phi:\ u_1 \xrightarrow{a,\gamma',\delta',\mathbf{x}'} u_1',\ \phi \Rightarrow \gamma',\ $ and $\ (t_1',u_1',\phi{\downarrow_{\mathbf{xx}'}}\Uparrow) \in S_1$

$\Rightarrow$ $\left\{\text{by Def. of } ||_B^\otimes\right\}$

$\exists\Phi : \Phi$ is a $(\phi_1 \wedge \gamma)$-partition :

$\forall\phi \in \Phi:\ (u_1,u_2) \xrightarrow{a,\gamma',\delta',\mathbf{x}'} (u_1',u_2),\ \phi \Rightarrow \gamma',$
and $\ (t_1',u_1',\phi{\downarrow_{\mathbf{xx}'}}\Uparrow) \in S_1$

$\Rightarrow$ {by (2.4) and since $\phi \Rightarrow \gamma'$ implies $(\phi \wedge \phi_2) \Rightarrow \gamma'$}

$\exists\Phi : \Phi$ is a $(\phi_1 \wedge \gamma)$-partition :

$\forall\phi \in \Phi:\ (u_1,u_2) \xrightarrow{a,\gamma',\delta',\mathbf{x}'} (u_1',u_2),\ (\phi \wedge \phi_2) \Rightarrow \gamma',$
and $\ ((t_1',t_2),(u_1',u_2),(\phi{\downarrow_{\mathbf{xx}'}}\Uparrow \wedge \phi_2)) \in S$

[Notice that clocks in $\mathbf{xx}'$ are not manipulated by automata T_1^2 and T_2^2 and hence irrelevant in ϕ_2. W.l.o.g. we therefore can assume that $\phi_2 \Rightarrow \bigwedge\{x \geq 0 \mid x \in \mathbf{xx}'\}$. Consequently $\phi_2{\downarrow_{\mathbf{xx}'}}\Uparrow = \phi_2$ since ϕ_2 is $\Uparrow$-closed. (†)]

$\Rightarrow$ {Fact 2.1.7 and observation}

$\exists\Phi : \Phi$ is a $(\phi_1 \wedge \gamma)$-partition :

$$\forall \phi \in \Phi : \; (u_1, u_2) \xrightarrow{a,\gamma',\delta',\mathbf{x}'} (u_1', u_2),\; (\phi \wedge \phi_2) \Rightarrow \gamma',$$
$$\text{and } ((t_1', t_2), (u_1', u_2), (\phi \wedge \phi_2){\downarrow_{\mathbf{x}\mathbf{x}'}}\Uparrow) \in S$$

$$\left[\; \text{Take } \Phi' = \{\phi \wedge \phi_2 \mid \phi \in \Phi\}. \text{ Then } \Phi' \text{ is a } (\phi_1 \wedge \phi_2 \wedge \gamma)\text{-partition.} \;\right]$$

$\Rightarrow$ {by observation, taking $\phi' = \phi \wedge \phi_2$}

$\exists \Phi' : \Phi'$ is a $(\phi_1 \wedge \phi_2 \wedge \gamma)$-partition :

$$\forall \phi' \in \Phi' : \; (u_1, u_2) \xrightarrow{a,\gamma',\delta',\mathbf{x}'} (u_1', u_2),\; \phi' \Rightarrow \gamma',$$
$$\text{and } ((t_1', t_2), (u_1', u_2), \phi'{\downarrow_{\mathbf{x}\mathbf{x}'}}\Uparrow) \in S$$

Case: $a \notin B$ with $t_2 \xrightarrow{a,\gamma,\delta,\mathbf{x}} t_2'$ and $t_1' = t_1$ Symmetric to the previous case.

Case: $a \in B$ with $t_1 \xrightarrow{a,\gamma_1,\delta_1,\mathbf{x}_1} t_1'$, $t_2 \xrightarrow{a,\gamma_2,\delta_2,\mathbf{x}_2} t_2'$, $\gamma \equiv \gamma_1 \wedge \gamma_2$, and $\delta \equiv (\delta_1, \gamma_1) \otimes (\delta_2, \gamma_2)$

$t_1 \xrightarrow{a,\gamma_1,\delta_1,\mathbf{x}_1} t_1'$ and $t_2 \xrightarrow{a,\gamma_2,\delta_2,\mathbf{x}_2} t_2'$

$\Rightarrow$ {by prop. 3 in Def. 2.9, since $(t_1, u_1, \phi_1) \in S_1$ and $(t_2, u_2, \phi_2) \in S_2$}

$\exists \Phi_1 : \Phi_1$ is a $(\phi_1 \wedge \gamma_1)$-partition :

$$\forall \phi_1' \in \Phi_1 : \; u_1 \xrightarrow{a,\gamma_1',\delta_1',\mathbf{x}_1'} u_1',\; \phi_1' \Rightarrow \gamma_1',\; \text{and } (t_1', u_1', \phi_1'{\downarrow_{\mathbf{x}_1\mathbf{x}_1'}}\Uparrow) \in S_1$$

and

$\exists \Phi_2 : \Phi_2$ is a $(\phi_2 \wedge \gamma_2)$-partition :

$$\forall \phi_2' \in \Phi_2 : \; u_2 \xrightarrow{a,\gamma_2',\delta_2',\mathbf{x}_2'} u_2',\; \phi_2' \Rightarrow \gamma_2',\; \text{and } (t_2', u_2', \phi_2'{\downarrow_{\mathbf{x}_2\mathbf{x}_2'}}\Uparrow) \in S_2$$

$\Rightarrow$ {logics and notation}

$\exists \Phi_1, \Phi_2 : \Phi_i$ is a $(\phi_i \wedge \gamma_i)$-partition, for $i = 1, 2$:

$\forall \phi_1' \in \Phi_1,\; \phi_2' \in \Phi_2$:

$$u_1 \xrightarrow{a,\gamma_1',\delta_1',\mathbf{x}_1'} u_1',\; \phi_1' \Rightarrow \gamma_1',\; (t_1', u_1', \phi_1'{\downarrow_{\mathbf{x}_1\mathbf{x}_1'}}\Uparrow) \in S_1,$$
$$u_2 \xrightarrow{a,\gamma_2',\delta_2',\mathbf{x}_2'} u_2',\; \phi_2' \Rightarrow \gamma_2',\; \text{and } (t_2', u_2', \phi_2'{\downarrow_{\mathbf{x}_2\mathbf{x}_2'}}\Uparrow) \in S_2$$

[$\phi_1' \Rightarrow \gamma_1'$ and $\phi_2' \Rightarrow \gamma_2'$ imply $(\phi_1' \wedge \phi_2') \Rightarrow (\gamma_1' \wedge \gamma_2')$.
Besides, $(\phi_1'{\downarrow_{\mathbf{x}_1\mathbf{x}_1'}}\Uparrow) \wedge (\phi_2'{\downarrow_{\mathbf{x}_2\mathbf{x}_2'}}\Uparrow) \Leftrightarrow (\phi_1' \wedge \phi_2'){\downarrow_{\mathbf{x}_1\mathbf{x}_2\mathbf{x}_1'\mathbf{x}_2'}}\Uparrow$ because of Fact 2.1.7 and by observation (†) in previous case since clocks in $\mathbf{x}_i\mathbf{x}_i'$ do not appear in ϕ_j' for $i \neq j$.]

$\Rightarrow$ {by def. of $||_B^{\otimes}$, (2.4), and observation, taking $\delta' \equiv (\delta_1', \gamma_1') \otimes (\delta_2', \gamma_2')$}

$\exists \Phi_1, \Phi_2 : \Phi_i$ is a $(\phi_i \wedge \gamma_i)$-partition, for $i = 1, 2$:
$\quad \forall \phi_1' \in \Phi_1,\ \phi_2' \in \Phi_2$:
$\qquad (u_1, u_2) \xrightarrow{a, \gamma_1' \wedge \gamma_2', \delta', \mathbf{x}_1' \mathbf{x}_2'} (u_1', u_2'),\ (\phi_1' \wedge \phi_2') \Rightarrow (\gamma_1' \wedge \gamma_2'),$
$\qquad$ and $((t_1', t_2'), (u_1', u_2'), (\phi_1' \wedge \phi_2')\downarrow_{\mathbf{x}_1 \mathbf{x}_2 \mathbf{x}_1' \mathbf{x}_2'}\Uparrow) \in S$

[Take $\Phi = \{\phi_1' \wedge \phi_2' \mid \phi_1' \in \Phi_1$ and $\phi_2' \in \Phi_2\}$. Notice that Φ is a $(\phi_1 \wedge \phi_2 \wedge \gamma_1 \wedge \gamma_2)$-partition.]

$\Rightarrow$ {by observation, taking $\phi' = \phi_1' \wedge \phi_2'$}
$\exists \Phi : \Phi$ is a $(\phi_1 \wedge \phi_2 \wedge \gamma_1 \wedge \gamma_2)$-partition :
$\quad \forall \phi' \in \Phi : (u_1, u_2) \xrightarrow{a, \gamma_1' \wedge \gamma_2', \delta', \mathbf{x}_1' \mathbf{x}_2'} (u_1', u_2'),\ \phi' \Rightarrow (\gamma_1' \wedge \gamma_2'),$
$\qquad$ and $((t_1', t_2'), (u_1', u_2'), \phi'\downarrow_{\mathbf{x}_1 \mathbf{x}_2 \mathbf{x}_1' \mathbf{x}_2'}\Uparrow) \in S$

Property 4: We have to show that

$$\phi_1 \wedge \phi_2 \Rightarrow (dl((t_1, t_2), A) \Leftrightarrow dl((u_1, u_2), A)) \textit{ for all } A \subseteq \mathcal{A}.$$

By Lemma 2.4, it suffices to prove that

$$(\phi_1 \wedge \phi_2) \Rightarrow (dl((t_1, t_2), \{a\}) \Leftrightarrow dl((u_1, u_2), \{a\})).$$

Therefore, the following calculations are under the hypothesis that $\phi_1 \wedge \phi_2$ holds. We consider two different cases.

case: $a \notin B$

$dl((t_1, t_2), \{a\})$
$\Leftrightarrow$ {def. of dl}
$\quad \bigvee\{\delta \mid (t_1, t_2) \xrightarrow{a, \gamma, \delta, \mathbf{x}} (t_1', t_2')$ for some $\gamma, \mathbf{x}, t_1', t_2'\}$
$\Leftrightarrow$ $\left\{\text{def. } ||_B^{\otimes} \text{ with } a \notin B\right\}$
$\quad \bigvee\{\delta \mid t_1 \xrightarrow{a, \gamma, \delta, \mathbf{x}} t_1'$ for some $\gamma, \mathbf{x}, t_1'\}$
$\quad \vee \bigvee\{\delta \mid t_2 \xrightarrow{a, \gamma, \delta, \mathbf{x}} t_2'$ for some $\gamma, \mathbf{x}, t_2'\}$
$\Leftrightarrow$ {def. of dl}
$\quad dl(t_1, \{a\}) \vee dl(t_2, \{a\})$

[By (2.4), $(t_i, u_i, \phi_i) \in S_i$ from which $\phi_i \Rightarrow (dl(t_i\{a\}) \Leftrightarrow dl(u_i, \{a\}))$ by prop. 4 in Def. 2.9, for $i = 1, 2$.]

$\Leftrightarrow$ {by observation, recalling that we assume $\phi_1 \wedge \phi_2$ holds}

$$
\begin{array}{l}
dl(u_1, \{a\}) \vee dl(u_2, \{a\}) \\
\Leftrightarrow \{\text{reasoning as before}\} \\
dl((u_1, u_2), \{a\})
\end{array}
$$

case: $a \in B$

$$
\begin{array}{l}
dl((t_1, t_2), \{a\}) \\
\Leftrightarrow \{\text{def. of } dl\} \\
\bigvee\{\delta \mid (t_1, t_2) \xrightarrow{a,\gamma,\delta,\mathbf{x}} (t_1', t_2') \text{ for some } \gamma, \mathbf{x}, t_1', t_2'\} \\
\Leftrightarrow \left\{\text{def. } ||_B^{\otimes} \text{ and } a \in B\right\} \\
\bigvee\{(\delta_1, \gamma_1) \otimes (\delta_2, \gamma_2) \mid t_1 \xrightarrow{a,\gamma_1,\delta_1,\mathbf{x_1}} t_1' \text{ and } t_2 \xrightarrow{a,\gamma_2,\delta_2,\mathbf{x_2}} t_2' \\
\qquad\qquad\qquad\qquad \text{for some } \mathbf{x_1}, \mathbf{x_2}, t_1', t_2'\} \\
\Leftrightarrow \{\text{change of notation and logic}\} \\
\bigvee_{t_1 \xrightarrow{a,\gamma_1,\delta_1,\mathbf{x_1}} t_1'} \; \bigvee_{t_2 \xrightarrow{a,\gamma_2,\delta_2,\mathbf{x_2}} t_2'} (\delta_1, \gamma_1) \otimes (\delta_2, \gamma_2) \\
\Leftrightarrow \{\otimes \text{ distributes w.r.t. } \vee\} \\
\left(\bigvee_{t_1 \xrightarrow{a,\gamma_1,\delta_1,\mathbf{x_1}} t_1'} \delta_1 \; , \; \bigvee_{t_1 \xrightarrow{a,\gamma_1,\delta_1,\mathbf{x_1}} t_1'} \gamma_1 \right) \otimes \left(\bigvee_{t_2 \xrightarrow{a,\gamma_2,\delta_2,\mathbf{x_2}} t_2'} \delta_2 \; , \; \bigvee_{t_2 \xrightarrow{a,\gamma_2,\delta_2,\mathbf{x_2}} t_2'} \gamma_2 \right) \\
\Leftrightarrow \{\text{def. of } dl \text{ and } gd\} \\
\big(dl(t_1, \{a\}), gd(t_1, a)\big) \otimes \big(dl(t_2, \{a\}), gd(t_2, a)\big)
\end{array}
$$

$$
\left[\begin{array}{l}
\text{By (2.4), } (t_i, u_i, \phi_i) \in S_i \text{ from which } \phi_i \Rightarrow (dl(t_i, \{a\}) \Leftrightarrow dl(u_i, \{a\})) \\
\text{by prop. 4 in Def. 2.9, and } \phi_i \Rightarrow (gd(t_i, a) \Leftrightarrow gd(u_i, a)) \text{ by} \\
\text{Lemma 2.5, for } i = 1, 2.
\end{array}\right]
$$

$$
\begin{array}{l}
\Leftrightarrow \left\{\begin{array}{l}\text{by observation and cond. 2 of } \otimes \text{, recalling that we assume} \\ \phi_1 \wedge \phi_2 \text{ holds}\end{array}\right\} \\
\big(dl(u_1, \{a\}), gd(u_1, a)\big) \otimes \big(dl(u_2, \{a\}), gd(u_2, a)\big) \\
\Leftrightarrow \{\text{reasoning as before}\} \\
dl((u_1, u_2), \{a\})
\end{array}
$$

□

Because of Corollary 2.1 and Theorem 2.2, $\sim^{\nabla}$ is also a congruence.

The next lemma is core to the proof that $\sim^{\nabla}$ is the coarsest congruence included in $\sim$. We notice that it does not use constraints 1, 2, and 3 imposed on $\otimes$. The lemma exhibits a test automaton T_t that distinguishes, modulo bisimulation, two automata that are not ∇-bisimilar. Automaton

T_t is built by adding extra actions in such a way that, when composed with an automaton T, the composition can mimic in the original semantics the behavior of T in the extended semantics. In fact, the extra actions are the same drop (∇_D) and undrop (Δ) actions of the extended semantics.

Definition 2.10 *The test automaton $T_t = (\mathcal{L}_t, \mathsf{l}^0{}_t, \mathcal{C}_t, \longrightarrow)$ is a TAD with*

- *set of locations $\mathcal{L}_t = \{s_D \mid D \subseteq \mathcal{A}\}$,*
- *initial location $\mathsf{l}^0{}_t = \{s_\varnothing\}$,*
- *set of clocks $\mathcal{C}_t = \varnothing$ and,*
- *for all $D, D' \subseteq \mathcal{A}$, $a \notin D$, define*
  - $s_D \xrightarrow{a,\boldsymbol{tt},\mathbf{0}_\delta,\varnothing} s_\varnothing$,
  - $s_D \xrightarrow{\nabla_{D'},\boldsymbol{tt},\boldsymbol{ff},\varnothing} s_{D\cup D'}$, *and*
  - $s_D \xrightarrow{\Delta,\boldsymbol{tt},\boldsymbol{ff},\varnothing} s_\varnothing$.

Lemma 2.6 *Let T_1 and T_2 be TADs with set of locations $\mathcal{L}_1$ and $\mathcal{L}_2$ respectively. Let T_t be a test automaton. Suppose that*

$$T_1 \,||^{\otimes}_{\mathcal{A}}\, T_t \sim T_2 \,||^{\otimes}_{\mathcal{A}}\, T_t.$$

Then,

$$R = \{((t_1, D)\rho_1, (t_2, D)\rho_2) \mid t_1 \in \mathcal{L}_1,\ t_2 \in \mathcal{L}_2,\ s_D \in \mathcal{L}_t, \\ \textit{and}\ \ (t_1, s_D)\rho_1 \sim (t_2, s_D)\rho_2\ \}$$

is a bisimulation relation that witnesses $T_1 \sim^\nabla T_2$.

Proof: The proof of the lemma is fairly straightforward except in the case of the delay transition. Notice that a delay transition from (t, D) is governed by satisfaction of $\neg dl(t, \mathcal{A} - D)$ (by $\mathsf{A2}_\nabla$) while in (t, s_D), it is governed by $tpc(t, s_D)$. To prove that both predicates are equivalent it is necessary that $(\mathbf{0}_\delta, \mathbf{tt})$ is neutral for $\otimes$. We show that relation

$$R = \{((t_1, D)\rho_1, (t_2, D)\rho_2) \mid t_1 \in \mathcal{L}_1,\ t_2 \in \mathcal{L}_2,\ s_D \in \mathcal{L}_t, \\ \text{and}\ \ (t_1, s_D)\rho_1 \sim (t_2, s_D)\rho_2\ \} \tag{2.5}$$

is a bisimulation that witness $T_1 \sim^\nabla T_2$. First notice that for all initial location t_1^0 of T_1 there is an initial location t_2^0 of T_2 such that $(t_1^0, s_\varnothing)(\mathcal{C}_1 \mapsto 0) \sim (t_2^0, s_\varnothing)(\mathcal{C}_2 \mapsto 0)$. Then $((t_1^0, \varnothing)(\mathcal{C}_1 \mapsto 0), (t_2^0, \varnothing)(\mathcal{C}_2 \mapsto 0)) \in R$. Similarly, we have that for all initial location t_2^0 of T_2 there is an initial location t_1^0 of T_1 such that $((t_1^0, \varnothing)(\mathcal{C}_1 \mapsto 0), (t_2^0, \varnothing)(\mathcal{C}_2 \mapsto 0)) \in R$. Then, provided R is a bisimulation, $T_1 \sim^\nabla T_2$.

Notice, besides, that R is symmetric by symmetry of $\sim$. We proceed to prove the transfer property by doing case analysis on the type of edge.

discrete transition:

$$
\begin{aligned}
& (t_1, D)\rho_1 \xrightarrow{a} (t_1', D')\rho_1' \\
\Rightarrow\ & \{\text{by } \mathbf{A1}_\nabla\} \\
& \quad \exists \gamma_1, \delta_1, \mathbf{x_1} :\ t_1 \xrightarrow{a,\gamma_1,\delta_1,\mathbf{x_1}} t_1',\ D' = \varnothing,\ \rho_1' = \rho_1\{\mathbf{x_1} := 0\}, \\
& \qquad \text{and } \rho_1 \models \gamma_1 \\
\Rightarrow\ & \left\{\text{by def. of } ||^{\otimes}_{\mathcal{A}} \text{ and def. of } T_t\right\} \\
& \quad \exists \gamma_1, \delta_1, \mathbf{x_1} :\ (t_1, s_D) \xrightarrow{\Delta,\mathbf{tt},\mathbf{ff},\varnothing} (t_1, s_\varnothing) \xrightarrow{a,\gamma_1,(\delta_1,\gamma_1)\otimes(\mathbf{0}_\delta,\mathbf{tt}),\mathbf{x_1}} (t_1', s_\varnothing) \\
& \qquad \text{and } \rho_1 \models \gamma_1 \\
\Rightarrow\ & \{\text{by } \mathbf{A1}\} \\
& \quad (t_1, s_D)\rho_1 \xrightarrow{\Delta} (t_1, s_\varnothing)\rho_1 \xrightarrow{a} (t_1', s_\varnothing)\rho_1\{\mathbf{x_1} := 0\} \\
\Rightarrow\ & \{(t_1, s_D)\rho_1 \sim (t_2, s_D)\rho_2\} \\
& \quad (t_2, s_D)\rho_2 \xrightarrow{\Delta} (t_2'', s_{D''})\rho_2'',\ (t_1, s_\varnothing)\rho_1 \sim (t_2'', s_{D''})\rho_2'', \\
& \quad \text{and } (t_1, s_\varnothing)\rho_1 \xrightarrow{a} (t_1', s_\varnothing)\rho_1\{\mathbf{x_1} := 0\} \\
\Rightarrow\ & \{(t_1, s_\varnothing)\rho_1 \sim (t_2'', s_{D''})\rho_2\} \\
& \quad (t_2, s_D)\rho_2 \xrightarrow{\Delta} (t_2'', s_{D''})\rho_2'' \xrightarrow{a} (t_2', s_{D'''})\rho_2', \\
& \quad \text{and } (t_1', s_\varnothing)\rho_1\{\mathbf{x_1} := 0\} \sim (t_2', s_{D'''})\rho_2' \\
\Rightarrow\ & \{\text{by } \mathbf{A1}_\nabla\} \\
& \quad \exists \gamma, \delta, \mathbf{x} :\ (t_2, s_D) \xrightarrow{\Delta,\gamma,\delta,\mathbf{x}} (t_2'', s_{D''}),\ \rho_2 \models \gamma,\ \rho_2'' = \rho_2\{\mathbf{x} := 0\}, \\
& \quad (t_2'', s_{D''})\rho_2\{\mathbf{x} := 0\} \xrightarrow{a} (t_2', s_{D'''})\rho_2', \\
& \quad \text{and } (t_1', s_\varnothing)\rho_1\{\mathbf{x_1} := 0\} \sim (t_2', s_{D'''})\rho_2' \\
\Rightarrow\ & \left\{\gamma = \mathbf{tt},\ \delta = \mathbf{ff},\ \mathbf{x} = \varnothing,\ D'' = \varnothing, \text{ and } t_2'' = t_2, \text{ by defs. of } ||^{\otimes}_{\mathcal{A}} \text{ and } T_t\right\} \\
& \quad (t_2, s_D) \xrightarrow{\Delta,\mathbf{tt},\mathbf{ff},\varnothing} (t_2, s_\varnothing), \\
& \quad (t_2, s_\varnothing)\rho_2 \xrightarrow{a} (t_2', s_{D'''})\rho_2',\ \text{ and } (t_1', s_\varnothing)\rho_1\{\mathbf{x_1} := 0\} \sim (t_2', s_{D'''})\rho_2' \\
\Rightarrow\ & \{\text{by } \mathbf{A1}_\nabla\} \\
& \quad \exists \gamma_2, \delta_2, \mathbf{x_2} :\ (t_2, s_\varnothing) \xrightarrow{a,\gamma_2,\delta_2,\mathbf{x_2}} (t_2', s_{D'''}),\ \rho_2' = \rho_2\{\mathbf{x_2} := 0\}, \\
& \qquad \rho_2 \models \gamma_2,\ \text{ and } (t_1', s_\varnothing)\rho_1\{\mathbf{x_1} := 0\} \sim (t_2', s_{D'''})\rho_2\{\mathbf{x_2} := 0\} \\
\Rightarrow\ & \left\{\text{by def. of } ||^{\otimes}_{\mathcal{A}} \text{ and def. of } T_t\right\} \\
& \quad \exists \gamma_2, \delta_2, \mathbf{x_2} :\ t_2 \xrightarrow{a,\gamma_2,\delta_2,\mathbf{x_2}} t_2',\ D''' = \varnothing,\ \rho_2 \models \gamma_2,\ \text{ and} \\
& \qquad (t_1', s_\varnothing)\rho_1\{\mathbf{x_1} := 0\} \sim (t_2', s_\varnothing)\rho_2\{\mathbf{x_2} := 0\} \\
\Rightarrow\ & \{\text{by } \mathbf{A1}_\nabla \text{ and (2.5)}\}
\end{aligned}
$$

$$(t_2, D)\rho_2 \xrightarrow{a} (t_2', \varnothing)\rho_2\{\mathbf{x_2} := 0\}, \text{ and}$$
$$(t_1', \varnothing)\rho_1\{\mathbf{x_1} := 0\} \sim (t_2', \varnothing)\rho_2\{\mathbf{x_2} := 0\}$$

delay transition: We first notice that

$$\neg dl(t, \mathcal{A} - D) = \neg \bigvee\{\delta \mid t \xrightarrow{a,\gamma,\delta,\mathbf{x}} t' \text{ and } a \in \mathcal{A} - D \text{ for some } \gamma, \mathbf{x}, t'\}$$

[Recall that $s_D \xrightarrow{a,\gamma',\delta',\mathbf{x}'} s'$ implies $a \in \mathcal{A} - D$, $\gamma' = \mathbf{tt}$, $\delta' = \mathbf{0}_\delta$, and $s' = s_\varnothing$, and that $(\mathbf{0}_\delta, \mathbf{tt})$ is neutral for $\otimes$. By def. of $||_{\mathcal{A}}^{\otimes}$ we obtain:]

$$\begin{aligned} &= \neg \bigvee\{\delta \mid (t, s_D) \xrightarrow{a,\gamma,\delta,\mathbf{x}} (t', s_\varnothing) \text{ for some } \gamma, \mathbf{x}, t'\} \\ &= tpc(t, s_D) \end{aligned} \tag{2.6}$$

Now we calculate:

$$\begin{aligned}
&(t_1, D)\rho_1 \xrightarrow{d} (t_1', D')\rho_1' \\
\Rightarrow\ &\{\text{by } \mathbf{A2}_\nabla\} \\
&\forall d' < d:\ \rho_1 + d' \models \neg dl(t_1, \mathcal{A} - D),\ t_1' = t_1,\ D' = D, \text{ and } \rho_1' = \rho_1 + d \\
\Rightarrow\ &\{\text{by (2.6)}\} \\
&\forall d' < d:\ \rho_1 + d' \models tpc(t_1, s_D) \text{ and } \rho_1' = \rho_1 + d \\
\Rightarrow\ &\{\text{by } \mathbf{A2}\} \\
&(t_1, s_D)\rho_1 \xrightarrow{d} (t_1, s_D)(\rho_1 + d) \\
\Rightarrow\ &\{(t_1, s_D)\rho_1 \sim (t_2, s_D)\rho_2\} \\
&(t_2, s_D)\rho_2 \xrightarrow{d} (t_2', s_{D''})\rho_2' \text{ and } (t_1, s_D)(\rho_1 + d) \sim (t_2', s_{D''})\rho_2' \\
\Rightarrow\ &\{\text{by } \mathbf{A2}\} \\
&\forall d' < d:\ \rho_2 + d' \models tpc(t_2, s_D),\ t_2' = t_2,\ D'' = D,\ \rho_2' = \rho_2 + d, \\
&\text{and } (t_1, s_D)(\rho_1 + d) \sim (t_2, s_D)(\rho_2 + d) \\
\Rightarrow\ &\{\text{by (2.6)}\} \\
&\forall d' < d:\ \rho_2 + d' \models \neg dl(t_2, D) \text{ and } (t_1, s_D)(\rho_1 + d) \sim (t_2, s_D)(\rho_2 + d) \\
\Rightarrow\ &\{\text{by } \mathbf{A2}_\nabla \text{ and (2.5)}\} \\
&(t_2, D)\rho_2 \xrightarrow{d} (t_2, D)(\rho_2 + d) \text{ and } (t_1, D)(\rho_1 + d) \sim (t_2, D)(\rho_2 + d)
\end{aligned}$$

drop transition:

$$\begin{aligned}
&(t_1, D)\rho_1 \xrightarrow{\nabla_E} (t_1', D')\rho_1' \\
\Rightarrow\ &\{\text{by } \mathbf{A3}\} \\
&D' = D \cup E,\ \rho_1' = \rho_1, \text{ and } t_1' = t_1 \\
\Rightarrow\ &\left\{\text{by def. of } ||_{\mathcal{A}}^{\otimes} \text{ and def. of } T_t\right\}
\end{aligned}$$

$$\begin{aligned}
&(t_1, s_D) \xrightarrow{\nabla_E,\mathbf{tt},\mathbf{ff},\varnothing} (t_1, s_{D\cup E})\\
\Rightarrow\ &\{\text{by } \mathbf{A2}\}\\
&(t_1, s_D)\rho_1 \xrightarrow{\nabla_E} (t_1, s_{D\cup E})\rho_1\\
\Rightarrow\ &\{(t_1, s_D)\rho_1 \sim (t_2, s_D)\rho_2\}\\
&(t_2, s_D)\rho_2 \xrightarrow{\nabla_E} (t_2', s_{D'})\rho_2' \text{ and } (t_1, s_{D\cup E})\rho_1 \sim (t_2', s_{D'})\rho_2'\\
\Rightarrow\ &\left\{\text{by } \mathbf{A2}\text{, def. of } ||_{\mathcal{A}}^{\otimes} \text{ and def. of } T_t\right\}\\
&(t_2, s_D) \xrightarrow{\nabla_E,\mathbf{tt},\mathbf{ff},\varnothing} (t_2, s_{D\cup E}),\ t_2' = t_2,\ D' = D\cup E,\ \rho_2' = \rho_2,\\
&\text{and } (t_1, s_{D\cup E})\rho_1 \sim (t_2, s_{D\cup E})\rho_2\\
\Rightarrow\ &\{\text{by } \mathbf{A3} \text{ and (2.5)}\}\\
&(t_2, D)\rho_2 \xrightarrow{\nabla_E} (t_2, D\cup E)\rho_2 \text{ and } (t_1, D\cup E)\rho_1 \sim (t_2, D\cup E)\rho_2
\end{aligned}$$

undrop transition:

$$\begin{aligned}
&(t_1, D)\rho_1 \xrightarrow{\Delta} (t_1', D')\rho_1'\\
\Rightarrow\ &\{\text{by } \mathbf{A4}\}\\
&D' = \varnothing,\ \rho_1' = \rho_1, \text{ and } t_1' = t_1\\
\Rightarrow\ &\left\{\text{by def. of } ||_{\mathcal{A}}^{\otimes} \text{ and def. of } T_t\right\}\\
&(t_1, s_D) \xrightarrow{\Delta,\mathbf{tt},\mathbf{ff},\varnothing} (t_1, s_\varnothing)\\
\Rightarrow\ &\{\text{by } \mathbf{A2}\}\\
&(t_1, s_D)\rho_1 \xrightarrow{\Delta} (t_1, s_\varnothing)\rho_1\\
\Rightarrow\ &\{(t_1, s_D)\rho_1 \sim (t_2, s_D)\rho_2\}\\
&(t_2, s_D)\rho_2 \xrightarrow{\Delta} (t_2', s_{D'})\rho_2' \text{ and } (t_1, s_\varnothing)\rho_1 \sim (t_2', s_{D'})\rho_2'\\
\Rightarrow\ &\left\{\text{by } \mathbf{A2}\text{, def. of } ||_{\mathcal{A}}^{\otimes} \text{ and def. of } T_t\right\}\\
&(t_2, s_D) \xrightarrow{\Delta,\mathbf{tt},\mathbf{ff},\varnothing} (t_2, s_\varnothing),\ t_2' = t_2,\ D' = \varnothing,\ \rho_2' = \rho_2,\\
&\text{and } (t_1, s_\varnothing)\rho_1 \sim (t_2, s_\varnothing)\rho_2\\
\Rightarrow\ &\{\text{by } \mathbf{A4} \text{ and (2.5)}\}\\
&(t_2, D)\rho_2 \xrightarrow{\Delta} (t_2, \varnothing)\rho_2 \text{ and } (t_1, \varnothing)\rho_1 \sim (t_2, \varnothing)\rho_2
\end{aligned}$$

□

From Lemma 2.6, it follows that $\sim^{\nabla}$ and $\sim^{\phi_0}$ are the coarsest congruence in $\sim$.

Theorem 2.3 *Fix $\otimes$ satisfying conditions 1 and 2 in Sec. 2.2. Then $\sim^{\nabla}$ (and hence $\sim^{\phi_0}$) is the coarsest congruence included in $\sim$ for the family of operators $||_B^{\otimes}$, with $B \subseteq \mathcal{A}$.*

Proof: Define $\simeq^{\otimes}$ to be the coarsest congruence for parallel composition contained in $\sim$, that is $T_1 \simeq^{\otimes} T_2 \Leftrightarrow \forall T, B: \ T_1 \,||_B^{\otimes}\, T \sim T_2 \,||_B^{\otimes}\, T$. We show that $\sim^{\nabla} = \sim^{\phi_0} = \simeq^{\otimes}$.

The fact that $\sim^{\nabla} = \sim^{\phi_0} \subseteq \simeq^{\otimes} \subseteq \sim$ follows from Lemma 2.1, Corollary 2.1, Theorem 2.2 and the fact that $\simeq^{\otimes}$ is the coarsest congruence included in $\sim$.

On the other direction, that is $\simeq^{\otimes} \subseteq \sim^{\phi_0}$, notice that $T_1 \simeq^{\otimes} T_2$ implies $T_1 \,||_{\mathcal{A}}^{\otimes}\, T_t \sim T_2 \,||_{\mathcal{A}}^{\otimes}\, T_t$ with T_t as in Lemma 2.6. Using Lemma 2.6 we can conclude that $T_1 \sim^{\nabla} T_2$. □

2.6 Concluding Remarks

Remark on Deciding ∇-bisimulation.

Our symbolic characterisation is based on [LY02] and [Čerāns92]. In particular, [Čerāns92] states that bisimulation is decidable for timed automata. The same applies to our relation. Since the number of regions is finite so is the number of (relevant) constraints (modulo logic equivalence) and as a consequence also the number of relevant $\Uparrow$-closed constraints. Therefore, any possible symbolic bisimulation relating two TADs will also be finite. Besides, operations $\downarrow_{\mathbf{x}}$ and $\Uparrow$ are expressible in terms of constraints, and it is possible to decide validity of the constraints on clocks. Following [Čerāns92], checking that two TADs T_1 and T_2 are ∇-bisimilarity is then possible by taking relation $S = \{(t, u, \phi\Uparrow) \mid \phi \in \mathcal{RC}(\mathbf{tt})\}$ (which is the finest partition possible since $\mathcal{RC}(\mathbf{tt})$ is the set of all regions) and checking that the transfer rules in Def. 2.9 hold for all tuples reachable from some set $I \subseteq (S \cap (\mathsf{ini}_1 \times \mathsf{ini}_2 \times \mathcal{RC}(\phi_0)))$ such that it relates all initial states of T_1 (resp. T_2) with some initial state of T_2, (resp. T_1).

Remark on Symbolic Bisimulation.

The third constraint in the definition of symbolic bisimulation (Def. 2.9) can be relaxed as follows:

> whenever $t \xrightarrow{a,\gamma,\delta,\mathbf{x}} t'$, there is a $(\phi\wedge\gamma)$-partition Φ s.t. for each $\phi'\in\Phi$, $u \xrightarrow{a,\gamma',\delta',\mathbf{y}} u'$, $\phi' \Rightarrow \gamma'$, $\phi'\downarrow_{\mathbf{xy}}\Uparrow \Rightarrow \psi$, and $(t', u', \psi) \in S$, for some ψ, γ', δ', $\mathbf{y}$ and u'.

the difference being in the existence of ψ such that $\phi'\downarrow_{\mathbf{xy}}\Uparrow \Rightarrow \psi$. It is not difficult to check that the new characterisation is equivalent to the original definition. This modification is important since it allows to obtain smaller relations due to the fact that a tuple $(t, u, \phi) \in S$ is redundant if there is a different tuple $(t, u, \phi') \in S$ such that $\phi \Rightarrow \phi'$.

Remark on Synchronising Constraints in Parallel Compositions.

In [BS00] the synchronisation of guards and deadlines of synchronising actions are defined by two operations which we call here $\oplus$ and $\otimes$ respectively. Some conditions are imposed on $\oplus$ and the only condition imposed on $\otimes$ is that $(\delta_1, \gamma_1) \otimes (\delta_2, \gamma_2) \Rightarrow (\gamma_1 \oplus \gamma_2)$ whenever $\delta_1 \Rightarrow \gamma_1$ and $\delta_2 \Rightarrow \gamma_2$ ([BS00] also suggest that $(\delta_1, \gamma_1) \otimes (\delta_2, \gamma_2) \Rightarrow (\delta_1 \vee \delta_2)$ should hold). We will only discuss here some particular examples that have recurred in the work of Sifakis et al. (see, e.g. [BST98, BS98a, BS00]). We first focus on the guard:

$\oplus = \wedge$. This is the one we use and amounts to checking that both guards are enabled in order to enable the synchronised transition.

$\oplus = \vee$. The synchronised transition can execute if any of the partners can do so.

$\oplus = \mathsf{max}$, where $\gamma_1 \mathsf{max} \gamma_2 = (\gamma_1 \wedge \gamma_2 \Uparrow) \vee (\gamma_2 \wedge \gamma_1 \Uparrow)$. In this case, a component is willing to synchronise if the synchronising transition was enabled in the past and the other component is ready to synchronise now.

$\oplus = \mathsf{min}$, where $\gamma_1 \mathsf{min} \gamma_2 = (\gamma_1 \wedge \gamma_2 \Downarrow) \vee (\gamma_2 \wedge \gamma_1 \Downarrow)$ with $\Downarrow$ being the *time predecessor* operator (the dual of $\Uparrow$). In this case, the synchronised guard anticipates the execution of the synchronising transitions.

Our congruence relation only works for $\wedge$. It is debatable how reasonable the other operations are. Synchronisation through $\vee$ is highly questionable. It is expected that automata T_{11} and T_{12} in Fig. 2.4 are equivalent under any reasonable criterion. Nevertheless, the composition $T_{11} \,||_a^{\otimes}\, T'''$ can perform action a at any moment while $T_{12} \,||_a^{\otimes}\, T'''$ cannot.

Under min, a component may anticipate the future behaviour of the synchronising partner. [BST98] and [BS00] suggest that the intention of this synchronisation is that the earliest synchronising transition makes irrelevant the second one (e.g. a tram leaves a crossing and after a while it signals to allow the change of the traffic light though it may be ignored if the light has already changed [BS00]). This intuition does not completely match the behaviour of min which will speed up the slower component allowing it to do activity otherwise impossible. This is observed when automaton T''' is composed with T_{13} and with T_{14} synchronising on a (see Fig. 2.4). Notice that T_{13} and T_{14} exhibit apparently equal behaviour since action a in T_{13} is always too late to execute b. However, the composition $T_{13} \,||_a^{\otimes}\, T'''$ may hasten the synchronisation on a making b apparent.

Dually, under max, an automaton may allow the execution of the synchronising action if it was enabled in the past. Notice that T_{15} and T_{16} in Fig. 2.4 exhibit equivalent behaviour: c cannot be executed in T_{15} since clock y is always set too early. Instead, the composition with T'''' synchronising on a will delay the execution long enough to set y sufficiently late to

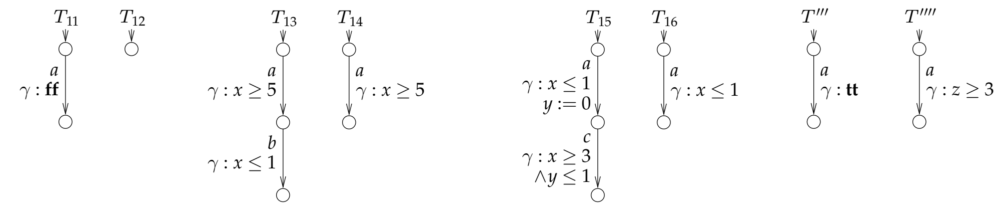

Figure 2.4: $T_{11} \sim^{\nabla} T_{12}$, $T_{13} \sim^{\nabla} T_{14}$, and $T_{15} \sim^{\nabla} T_{16}$

enable the c transition. The intention behind this form of synchronisation is that the fastest component can always wait for the slowest. This design choice seems an adequate choice to use with soft deadlines. Notice also that the appearance of new activity is reasonable since it may be important to cope with the occasional delay. What is debatable is the need of max since this type of synchronisation can easily be represented using $\wedge$: Notice that the max synchronisation does not allow any test automata to distinguish between γ and $\gamma \Uparrow$. Hence, it is more reasonable to model this kind of synchronisation using $\wedge$ instead of max and let all guards be $\Uparrow$-closed.

With respect to deadlines, [BS00] is more liberal. The two type of synchronising deadlines that stand out are:

Patient synchronisation: $(\delta_1, \gamma_1) \otimes (\delta_2, \gamma_2) = \delta_1 \wedge \delta_2$ with $\mathbf{0}_\delta = \mathbf{tt}$, and

Impatient synchronisation: $(\delta_1, \gamma_1) \otimes (\delta_2, \gamma_2) = (\delta_1 \vee \delta_2) \wedge (\gamma_1 \wedge \gamma_2)$ with $\mathbf{0}_\delta = \mathbf{ff}$.

The nomenclature corresponds to [DHKK01] but these definitions are already introduced in [SY96] with the names of *flexible* and *stiff* respectively. Patient synchronisation allows to model soft deadlines, in the sense that one of the components is always willing to wait for the other (as long as its guards remain valid). On the other hand, impatient synchronisation imposes urgency and obliges the execution as soon as both partners are ready to execute the synchronising transition. Both [SY96] and [DHKK01, BDHK04] give a weaker definition of impatient synchronisation: $(\delta_1, \gamma_1) \otimes (\delta_2, \gamma_2) = \delta_1 \vee \delta_2$. Taking $\mathbf{0}_\delta = \mathbf{ff}$, our result is also valid for this definition. The only problem with it is that it does not preserve time reactivity, i.e. condition 1 on $\otimes$ (see Sec. 2.2) does not hold[2].

We finally mention that ∇-bisimulation is still a congruence for $||_B^{\otimes}$ if condition 4 on $\otimes$ is dropped. However, it is *not* the coarsest congruence in $\sim$ any longer. (This can easily be seen by taking $(\delta_1, \gamma_1) \otimes (\delta_2, \gamma_2) = \mathbf{ff}$).

[2]To strictly model hard deadlines, this composition requires some modification of the rules in order to ensure the time-blockage produced when a component is ready to synchronise but the other cannot do it at all. A possible solution appears in [BDHK04].

Conclusion

We have characterised the coarsest congruence for parallel compositions of TADs with soft and hard deadline synchronisation that is included in bisimulation. We also gave a symbolic characterisation of it and show that it is decidable. The novelty in our result is that the proof of congruence was entirely carried out in the symbolic semantics rather than resorting to the underlying transition system. The choice of this strategy is not fortuitous. It is mainly due to the complexity of defining an equivalent parallel composition on transition systems. To begin with, any possible definition needs to be tailored for a particular choice of deadline. Besides, it would need complex bookkeeping to know which possible deadline is blocking the passage of time. Many other different complications appear depending on the choice of $\otimes$.

We finally discussed different types of synchronisation in parallel composition and conclude that our choice is both reasonable and sufficiently expressive to consider the modelling of both soft and hard real-time constraints.

Chapter 3

Axiomatization of Timed Automata with Deadlines

with Pedro R. D'Argenio

Abstract It is known that the usual timed bisimulation fails to be a congruence for timed automata with deadlines – a variant of timed automata where component synchronization is delayable, and time progress is controlled by deadlines on transitions instead of invariants on locations. In the previous chapter we presented the ∇-bisimulation which is the coarsest congruence relation that is included in timed bisimulation for timed automata with deadlines. In the present chapter we provide an algebraic proof system for direct derivation of such a relation by syntactic manipulation. In the squel, we establish that the proof system is sound and complete.

3.1 Introduction

We begin by defining a CCS-style language [Mil89a] (denoted by $\mathbb{A}$) to represent TAD algebraically. The language $\mathbb{A}$ shall have clock guard and deadline constructs to express enabling and enforcing conditions, action prefix, clock resetting and recursive constructs. Semantically, $\mathbb{A}$ is equipped with transitional semantics in terms of transition systems and symbolic transitions in terms of TADs. $\mathbb{A}$ has a proof system with axioms and inference rules to mimic the ∇-bisimulation of TADs. Similar to ∇-bisimulation ($\sim^{\nabla}$) and symbolic bisimulation ($\sim^{\phi}$) relations of TADs, the equivalence relation in $\mathbb{A}$ is subject to clock values. Hence the equivalence relation of $\mathbb{A}$ is a conditional equation of the form

$$\phi \vdash t = u$$

where, ϕ is a clock constraint, and t and u are terms in $\mathbb{A}$. To prove the equivalence of $=$ and $\sim^{\nabla}$(or $\sim^{\phi}$), we shall show that the proof system is sound and complete. The soundness proof follows a similar approach as in [LY02]. That is, we shall define an intermediate but equivalent bisimulation relation up to d, which helps to simplify the proof. The completeness proof follows the arguments used by Milner [Mil84, Mil89b]. This approach is also used in [AJ95, LY02]. The proof is divided into three parts. The first part transforms the set of equations to a special equation called *standard equations*. The second and third parts of the proof deals with the completeness of guarded and unguarded equations, respectivily.

Axiomatizations of timed automata have already appeared in [DB96] and [LY02]. The former one presents a sound axiomatization for safe timed automata [HNSY94]. The latter one presents a sound and complete proof system for bisimulation in the same class of automata. Our work is closely related to this one, but is focused on a different model and a different type of bisimulation. Apart from the different setting, the following new results (w.r.t. [LY02]) are given: First, our algebra has only one sort (in [LY02], the algebra contains two sorts —one with invariants and the other without). On one hand, a one sort language is simpler and more general, on the other hand the proof system became difficult, in particular defining the time progess condition $dl(t, A)$ (Fig. 3.1) and the transformation of equations to standard equations in the completness proof (Lemma 3.3) is complicated as a result of the generalized language. The second result is a completeness result also for unguarded recursion.

The chapter is organized as follows. Section 3.2 defines a CCS-style language to describe TADs. Its semantics and bisimulation relation is defined at the end of this section. Section 3.3 contains the axioms and inference rules of the language, followed by Section 3.4 discussing some useful properties of the proof system. Section 3.5 proves the soundness of the proof system, while Sections 3.6.2 and 3.6.3 prove completeness for guarded and unguarded terms, respectively. Most of the symbols and terminology used in this chapter are already defined in the previous chapter. Unless explicitly stated, they retain their original meaning.

3.2 Algebra for Timed Automata with Deadlines

Let $\mathcal{A}$ be a finite set of actions, ranged over by a, b. Let $\mathbf{X}$ be a set of process variables ranged over by X, Y, and let $\gamma, \delta \in \mathcal{F}(\mathcal{C})$ be clock constraints. The *Algebra for Timed Automata with Deadlines* $\mathbb{A}$ over $\mathcal{A}, \mathcal{C}$ and $\mathbf{X}$ is given by the following BNF grammar:

$$t \quad ::= \quad \mathbf{0} \mid \gamma \rightarrow t \mid \delta : t \mid t + t \mid \mathbf{fix} X\, t \mid a(\mathbf{x}).t \mid X \tag{3.1}$$

The expression $a(\mathbf{x}).t$ with $a \in \mathcal{A}$ is the action prefixing operator with clock resetting. The clock constraints γ and δ are called *guard* and *deadline* constraints, respectively. The term $\gamma \rightarrow t$ represents a conditional construction such that when the guard γ is true, it *may* perform any action t is able to perform. The term $\delta : t$ represents a deadline construction such that when the deadline δ is true, the process *must* perform some action that t can perform. We assume δ is left closed. The term $\mathbf{0}$ denotes an inactive process which can do nothing except allowing time to pass. The process $\mathbf{tt} : t$ behaves the same as t except it forces the execution of any enabled action before letting time pass. We call $\mathbf{tt} : t$ an *urgent* process. As usual, $t_1 + t_2$ and $\mathbf{fix}Xt$ are, respectively, the non-deterministic choice and the recursion operation.

We say that a variable X occurs *unguarded* in a term t if such an occurrence is not within the scope of an action prefix. If X does not occur unguarded in t we say that X is *guarded* in t. Hence, X occurs unguarded in $(x \geq 5) : ((x \geq 2) \rightarrow X + a(\{y\}).Y)$, but Y is guarded. (Note that that the concept of guarded variable is *not* related the guard operation.) A term t is *guarded* if all of its subterms of the form $\mathbf{fix}Xu$ and X is guarded in u.

To reduce the number of parenthesis we, adopt the following binding power in decreasing order on the operators: action prefix, $\mathbf{fix}X$, deadline, guard and summation.

Example 3.1 *Consider the following simple* `ssh` *server login procedure. Initially the server is idle, until a client program requests a connection via action* a. *The server accepts the request and it waits for 2 minutes for the user to enter his/her user name and password. If this is achieved the server passes control to a login verifier via action* b. *If the user name and password matches (action* e*) the user enters the system. Otherwise, the server loops back (action* d*) and asks the user to enter his/her user name and password again. After waiting for 2 minutes if no user name and password is entered, the connection is broken (action* c*) and the server is back to its idle state. This can be modeled in* $\mathbb{A}$, *using one clock variable* x, *by the process* `ssh` *below*

$$\begin{array}{rcl} \texttt{ssh} & \equiv & s_0 \\ s_0 & \equiv & \mathbf{fix}X_0(a(\{x\}).s_1) \\ s_1 & \equiv & \mathbf{fix}X_1((x{=}2) : (x{\leq}2 \rightarrow b(\varnothing).s_2 + x{\geq}2 \rightarrow c(\varnothing).X_0)) \\ s_2 & \equiv & d(\{x\}).X_1 + e(\varnothing).\mathbf{0} \end{array}$$

3.2.1 Transitional Semantics

The semantics of $\mathbb{A}$ is formally defined in terms of a timed transitions system $TS_{\nabla} = (\mathcal{S}, \Sigma, \longrightarrow)$ where

- $\mathcal{S} \subseteq (\mathbb{A} \times 2^{\mathcal{A}}) \times (\mathcal{C} \rightarrow \mathbb{R}_{\geq 0})$ is set of states
- $\Sigma = \mathcal{A} \cup \mathbb{R}_{\geq 0} \cup \mathcal{A}_{\nabla} \cup \{\Delta\}$ is set of vocabulary, where $\mathcal{A}_{\nabla} = \{\nabla_A \mid A \subseteq \mathcal{A}\}$ and Δ are the drop and undrop actions as described in the previous chapter.
- $\longrightarrow$ (the transition relation) is defined as in Fig. 3.2.

$$
\begin{array}{ll}
dl(t_0 + t_1, A) = dl(t_0, A) \vee dl(t_1, A) & I(t_0 + t_1) = I(t_0) \cup I(t_1) \\
dl(\mathbf{fix}Xt, A) = dl(t[\mathbf{fix}Xt/X], A) & I(\mathbf{fix}Xt) = I(t[\mathbf{fix}Xt/X]) \\
dl(\delta : t, A) = \begin{cases} (\delta \wedge gd(t, A)) \vee dl(t, A) & \text{if } A \cap I(t) \neq \varnothing \\ \mathbf{ff} & \text{otherwise} \end{cases} & \begin{array}{l} I(\delta : t) = I(t) \\ I(\gamma \to t) = I(t) \end{array} \\
dl(\gamma \to t, A) = \gamma \wedge dl(t, A) & I(a(\mathbf{x}).t) = \{a\} \\
dl(a(\mathbf{x}).t, A) = dl(\mathbf{0}, A) = dl(X, A) = \mathbf{ff} & I(\mathbf{0}) = I(X) = \varnothing
\end{array}
$$

$$
\begin{array}{ll}
gd(\mathbf{0}, A) = \mathbf{ff} \quad gd(a(\mathbf{x}).t, A) = gd(X, A) = \mathbf{tt} & gd(t + u, A) = gd(t, A) \vee gd(u, A) \\
gd(\gamma \to t, A) = \begin{cases} \gamma \wedge gd(t, A) & \text{if } A \cap I(t) \neq \varnothing \\ \mathbf{ff} & \text{otherwise} \end{cases} & \begin{array}{l} gd(\mathbf{fix}Xt, A) = gd(t[\mathbf{fix}Xt/X], A) \\ gd(\delta : t, A) = gd(t, A) \end{array}
\end{array}
$$

Figure 3.1: Definitions of deadline (*dl*) and set of initial actions (I)

The transition relation ($\longrightarrow$) is defined using eight rules in Fig. 3.2. Most of the rules are fairly obvious except the DELAY rule. The DELAY rule uses the predicate $dl(t, A)$, which ensures the maximum delay time (or as commonly know as the *time progress condition*) of the term t. We shall first explain the definition of $dl(t, A)$ given in Fig. 3.1, and then come back to the definition of the transition relation.

Given a set of active (undropped) actions A, the maximum delay time of a term t is based on the following two conditions.

1. The set of all actions such that a sub term $a(\mathbf{x}).u$ in t occurs out of the scope of another action prefix is denoted by $I(t)$ which is also formally defined as the smallest set satisfying equations in Fig. 3.1. The set $A \cap I(t)$ contains all active actions whose deadline is going to be considered in $dl(t, A)$.

2. To guarantee *timelock freedom*, we only consider deadlines that imply the guard of an action they refer to. Let $gd(t, A)$ be the enabling condition in t of all actions in A, i.e. $gd(t, A)$ is satisfied in valuation ρ iff t can perform some actions $a \in A$ in ρ. Formally, it is defined as the weakest predicate satisfying equations in Fig. 3.1. To this end, every deadline δ is conjugated with $gd(t, A)$ as shown in Fig. 3.1.

The deadline of a term t considering only deadlines on actions in $A \subseteq \mathcal{A}$ is the disjunction of all deadlines imposed on any enabled action $a \in A \cap I(t)$ and originating from t. Formally, $dl(t, A)$ is defined as the weakest predicate satisfying equations in Fig. 3.1.

The semantics of $\mathbb{A}$ is given in Fig. 3.2 in a structural way. The rule GUARD allows the execution of an action only if guard γ is valid in the current valuation. DEADLINE states that deadlines have no effect on discrete actions. The rule ACTION allows the execution of discrete action and clock reset. DELAY defines the time progress: a state $(t, D)\rho$ can progress d time units if no deadline under consideration is reached within the period in which $dl(t, \mathcal{A}-D)$ is false. Rules DROP and UNDROP define the effect of

$$\text{DELAY}\quad \frac{\forall d' < d \quad \rho + d' \models \neg dl(t, \mathcal{A} - D)}{(t, D)\rho \xrightarrow{d} (t, D)\rho + d} \qquad \text{REC}\quad \frac{(t[\mathbf{fix}Xt/X], D)\rho \xrightarrow{a} (t', D')\rho'}{(\mathbf{fix}Xt, D)\rho \xrightarrow{a} (t', D')\rho'}$$

$$\text{ACTION}\quad \frac{}{(a(\mathbf{x}).t, D)\rho \xrightarrow{a} (t, \varnothing)\rho\{\mathbf{x}{:=}0\}} \qquad \text{DROP}\quad \frac{}{(t, D)\rho \xrightarrow{\nabla_A} (t, D \cup A)\rho}$$

$$\text{GUARD}\quad \frac{(t, D)\rho \xrightarrow{a} (t', D')\rho' \quad \rho \models \gamma}{(\gamma \to t, D)\rho \xrightarrow{a} (t', D')\rho'} \qquad \text{UNDROP}\quad \frac{}{(t, D)\rho \xrightarrow{\Delta} (t, \varnothing)\rho}$$

$$\text{DEADLINE}\quad \frac{(t, D)\rho \xrightarrow{a} (t', D')\rho'}{(\delta{:}t, D)\rho \xrightarrow{a} (t', D')\rho'} \qquad \text{CHOICE}\quad \frac{(t, D)\rho \xrightarrow{a} (t', D')\rho'}{\begin{array}{c}(t + u, D)\rho \xrightarrow{a} (t', D')\rho' \\ (u + t, D)\rho \xrightarrow{a} (t', D')\rho'\end{array}}$$

Figure 3.2: Transitional Semantics of $\mathbb{A}$

$$\text{ACTION}\quad \frac{}{a(\mathbf{x}).t \xrightarrow{a,\mathbf{tt},\mathbf{ff},\mathbf{x}} t} \qquad \text{CHOICE}\quad \frac{t \xrightarrow{a,\gamma,\delta,\mathbf{x}} t'}{\begin{array}{c}(t + u) \xrightarrow{a,\gamma,\delta,\mathbf{x}} t' \\ (u + t) \xrightarrow{a,\gamma,\delta,\mathbf{x}} t'\end{array}}$$

$$\text{REC}\quad \frac{t[\mathbf{fix}Xt/X] \xrightarrow{a,\gamma,\delta,\mathbf{x}} t'}{\mathbf{fix}Xt \xrightarrow{a,\gamma,\delta,\mathbf{x}} t'}$$

$$\text{GUARD}\quad \frac{t \xrightarrow{a,\gamma',\delta',\mathbf{x}} t'}{\gamma \to t \xrightarrow{a,\gamma\wedge\gamma',\delta'\wedge\gamma,\mathbf{x}} t'} \qquad \text{DEADLINE}\quad \frac{t \xrightarrow{a,\gamma',\delta',\mathbf{x}} t'}{\delta : t \xrightarrow{a,\gamma',\delta'\vee(\delta\wedge\gamma'),\mathbf{x}} t'}$$

Figure 3.3: Symbolic semantics of $\mathbb{A}$

the drop and undrop actions respectively. Note that they can be performed unconditionally.

The notion of equivalence underlying the algebra $\mathbb{A}$ is defined as follows. We say that two states $p = (t, D)\rho$ and $q = (u, E)\eta$ are ∇-*bisimilar*, notation $p \sim^\nabla q$ iff there exists a symmetric relation R (called ∇-*bisimulation*) such that for any $(p, q) \in R$ and $l \in \mathbb{R}_{\geq 0} \cup \mathcal{A} \cup \mathcal{A}_\nabla \cup \{\Delta\}$, whenever $p \xrightarrow{l} p'$ then $\exists q' : q \xrightarrow{l} q'$ and $p'Rq'$. If $p'Rq'$ is changed to $p' \sim^\nabla \circ R \circ \sim^\nabla q'$, R is a ∇-*bisimulation up to* $\sim^\nabla$ ($\circ$ is the usual composition on relations). It is enough to prove the existence of a ∇-bisimulation up to $\sim^\nabla$ between p and q to state that they are ∇-bisimilar [Mil89a].

3.2.2 Symbolic Semantics.

The symbolic semantics of an $\mathbb{A}$ term t in terms of TAD is defined by $T_t = (\mathcal{A}, t, \mathcal{C}, \twoheadrightarrow)$, where $\twoheadrightarrow$ is the smallest relation satisfying the rules in Fig. 3.3 and $\mathcal{A}$ is the set of actions for TADs.

Example 3.2 *The* $\mathbb{A}$ *model of the* `ssh` *server in Example 3.1 can be interpreted in terms of a TAD using the rules in Fig. 3.3. The resulting TAD is given in Fig. 3.4. The derivation is straightforward, the interesting part is how to "push"*

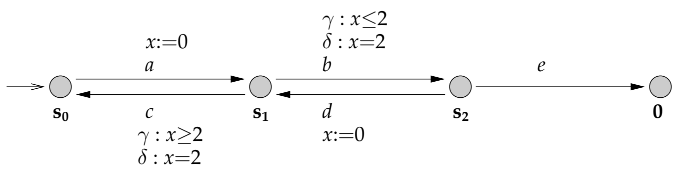

Figure 3.4: TAD for `ssh` login procedure

the deadline (x=2) *to both branches of* s_1. *This is done by applying* ACTION, GUARD, CHOICE, *and* DEADLINE *sequentially to both branches of* s_1.

Conversely, for a given TAD $T = (\mathcal{L}, \mathsf{l}^0, \mathcal{C}, \longrightarrow)$, its equivalent $\mathbb{A}$ term can be derived as follows. Suppose $\mathcal{L} = \{l_0, l_1, \ldots, l_n\}$ with $l_i \leq l_j$ iff $i \leq j$. For each $l \in \mathcal{L}$, let $J_l = \{e \mid e = (l, a_e, \gamma_e, \delta_e, \mathbf{x}_e, l_e) \in \longrightarrow\}$ and define

$$t_l \overset{\text{def}}{=} \mathbf{fix} X_l \Big(\textstyle\sum_{e \in J_l} \left(\gamma_e \to \delta_e : a_e(\mathbf{x}_e).u_{l_e} \right) \Big)$$

where

$$u_{l_e} = \begin{cases} X_{l_e} & \text{if } l_e \leq l \\ t_{l_e} & \text{otherwise} \end{cases}$$

Example 3.3 *t_{s_0} is the $\mathbb{A}$ term associated to the TAD of Fig. 3.4, where:*

$$\begin{array}{lcl} t_{s_0} & \equiv & \mathbf{fix} X_{s_0} (\mathbf{\mathit{ff}} : \mathbf{\mathit{tt}} \to a(\{x\}).t_{s_1}) \\ t_{s_1} & \equiv & \mathbf{fix} X_{s_1} ((x{=}2) : (x{\leq}2) \to b(\varnothing).t_{s_2} + (x{=}2) : (x{\geq}2) \to c(\varnothing).X_{s_0})) \\ t_{s_2} & \equiv & \mathbf{fix} X_{s_2} (\mathbf{\mathit{ff}} : \mathbf{\mathit{tt}} \to d(\{x\}).X_{s_1} + \mathbf{\mathit{ff}} : \mathbf{\mathit{tt}} \to e(\varnothing).t_{s_3}) \\ t_{s_3} & \equiv & \mathbf{fix} X_{s_3} \mathbf{0} \end{array}$$

The semantics of T in terms of transition systems is given by the transitional semantics of t_{l_0}. It is routine to show that the semantics of T given in this manner is the same as the one defined in the previous chapter. Moreover, by induction on the proof tree, it is also possible to show that the transitional semantics of t is the same as the two step semantics of t (i.e. interpret t as a TAD T_t and then obtain the transition system from T_t) provided that t is closed (i.e., it does not contain a variable X out of the scope of a $\mathbf{fix}X$.)

3.3 The Proof System

The proof system of $\mathbb{A}$ is given by the set of axioms and inference rules in Fig. 3.5 and Fig. 3.6 respectively. The judgments of the inference system are conditional equations of the form

$$\phi \vdash t = u$$

S1	$t + \mathbf{0} = t$	U1	$\mathbf{tt} : \mathbf{tt} : t = \mathbf{tt} : t$
S2	$t + t = t$	U2	$\mathbf{tt} : \delta : t = \delta : \mathbf{tt} : t$
S3	$(t + u) + v = t + (u + v)$	U3	$\mathbf{tt} : \gamma \rightarrow t = \gamma \rightarrow \mathbf{tt} : t$
S4	$t + u = u + t$	U4	$\mathbf{tt} : (t_1 + t_2) = \mathbf{tt} : t_1 + \mathbf{tt} : t_2$
		U5	$\mathbf{tt} : t = \mathbf{tt} : t + t$

DL	$(\delta \wedge \gamma_1) : (\gamma_1 \rightarrow a(\mathbf{x}).t + \gamma_2 \rightarrow a(\mathbf{y}).u) = \delta : \gamma_1 \rightarrow a(\mathbf{x}).t + \gamma_2 \rightarrow a(\mathbf{y}).u$
UR	$\mathbf{fix} X(t + \delta : \gamma \rightarrow X) = \mathbf{fix} X(t + \delta : \gamma \rightarrow t)$

Figure 3.5: The equational axioms

where ϕ is a constraint and t, u are terms. Its intended meaning is: t is equivalent to u whenever ϕ holds. We will abbreviate $\mathbf{tt} \vdash t = u$ as $t = u$ and consider in general two logically equivalent constraints as the same constraint (hence, e.g., $\mathbf{tt} \vdash t = u$ and $(x{+}1 \geq x) \vdash t = u$ are the same judgment.)

Axioms S1-4 are standard summation laws and U1-5 are axioms to manipulate urgent processes. Axiom UR explains in which way unguarded variables in recursion are redundant (notice the difference with Milner's [Mil89a] recursion axiom $\mathbf{fix} X(t + X) = \mathbf{fix} X t$). Axiom DL shows a particularity of the ∇-bisimulation: a deadline on an action has the same impact on another process as long as it is prefixed with the *same action*. Deadlines *cannot* be shifted out of any arbitrary summation. As a simple example, the term $\delta : a.(\mathbf{x}).t + b.(\mathbf{y}).u$ and $\delta : (a.(\mathbf{x}).t + b.(\mathbf{y}).u)$ will only be equivalent if and only if $a = b$. This is precisely what usual bisimulation would allow hence failing to be a congruence.

Each construct in the language has an entry in the set of inference rule of Fig. 3.6. They show how to use the constructs, and what constraints must be met (if any) before applying the rule. Three additional rules, namely, ABSURD, PARTITION and CONSEQUENCE are also given. They are used to manipulate the condition under which the equation holds. SUBSTITUTION rules handle substitution in the context of the choice operator and urgency. Rule ACTION is for action prefix with clock resetting. Informally, it states that the equivalence $a(\mathbf{x}).t = a(\mathbf{x}).t$ under ϕ can be inferred provided that $t = u$ under $\phi{\downarrow_{\mathbf{x}}}{\Uparrow}$. The clock constraint $\phi{\downarrow_{\mathbf{x}}}{\Uparrow}$ is obtained from ϕ by first setting the clocks in $\mathbf{x}$ to zero and then removing upper bounds on all clocks. GUARD does a case analysis on conditions, i.e. if

1. t behaves like u when the guard γ holds under ϕ (i.e., when $\phi \wedge \gamma$ holds), and

2. t behaves like $\mathbf{0}$ if this is not the case,

then $\phi \vdash \gamma \rightarrow t = u$ can be inferred. The rule DEADLINE is similar to GUARD except that t is required to be urgent when $\phi \wedge \delta$ holds. THINNING states

EQUIV	$\dfrac{}{t = t} \quad \dfrac{\phi \vdash t = u}{\phi \vdash u = t} \quad \dfrac{\phi \vdash t = u \quad \phi \vdash u = v}{\phi \vdash t = v}$
AXIOM	$\dfrac{}{t = u}$ $t = u$ an axiom instance
SUBSTITUTION	$\dfrac{\phi \vdash t = t'}{\phi \vdash t + u = t' + u} \quad \dfrac{\phi \vdash t = u}{\phi \vdash \mathbf{tt} : t = \mathbf{tt} : u}$
ACTION	$\dfrac{\phi{\downarrow_{\mathbf{x}}}{\Uparrow} \vdash t = u}{\phi \vdash a(\mathbf{x}).t = a(\mathbf{x}).u}$
GUARD	$\dfrac{\phi \wedge \gamma \vdash t = u \quad \phi \wedge \neg\gamma \vdash \mathbf{0} = u}{\phi \vdash \gamma \rightarrow t = u}$
DEADLINE	$\dfrac{\phi \wedge \neg\delta \vdash t = u \quad \phi \wedge \delta \vdash \mathbf{tt}{:}t = u}{\phi \vdash \delta{:}t = u}$
THINNING	$\dfrac{}{\phi \vdash a(\mathbf{xy}).t = a(\mathbf{x}).t} \quad \mathbf{y} \cap \mathcal{C}(t) = \varnothing$
REC	$\dfrac{}{\mathbf{fix}Xt = t[\mathbf{fix}Xt/X]}$
UFI	$\dfrac{t = u[t/X]}{t = \mathbf{fix}Xu}$
PARTITION	$\dfrac{\phi_1 \vdash t = u \quad \phi_2 \vdash t = u}{\phi_1 \vee \phi_2 \vdash t = u}$
CONSEQUENCE	$\dfrac{\psi \vdash t = u}{\phi \vdash t = u} \quad \phi \Rightarrow \psi$
ABSURD	$\dfrac{}{\mathbf{ff} \vdash t = u}$

Figure 3.6: The inference rules

that clocks which are not free in t (denoted by $\mathcal{C}(t)$) are redundant in a reset set of a prefix of t. There are two rules for recursion: REC is for folding or unfolding recursion expressions, while UFI states the uniqueness of the solution of recursive equations provided that the variable of interest only occurs guarded.

3.4 Properties of the Proof System

This section presents some selected properties of the proof system, which are used to prove the soundness and completeness theorems. Readers only interested in the soundness and completeness of the proof system can safely skip this (rather technical) section when reading for the first time.

Lemma 3.1 1. *If $\phi \Rightarrow \gamma$ and $\phi \vdash t = u$ then $\phi \vdash \gamma \rightarrow t = u$.*

2. *If $\phi \Rightarrow \neg\delta$ and $\phi \vdash t = u$ then $\phi \vdash \delta : t = u$.*

3. *If $\phi \Rightarrow \gamma \wedge \neg\delta$ and $\phi \vdash t = u$ then $\phi \vdash \gamma \rightarrow \delta : t = u$ and $\phi \vdash \delta : \gamma \rightarrow t = u$.*
4. $t = t + \phi \rightarrow t$.
5. $\gamma_1 \rightarrow \gamma_2 \rightarrow t = (\gamma_1 \wedge \gamma_2) \rightarrow t$
6. $\gamma \rightarrow (t_1 + t_2) = \gamma \rightarrow t_1 + \gamma \rightarrow t_2$.
7. $\gamma_1 \rightarrow t + \gamma_2 \rightarrow t = (\gamma_1 \vee \gamma_2) \rightarrow t$.
8. $\delta_1 : \delta_2 : t = (\delta_1 \vee \delta_2) : t$.
9. $\delta : (t_1 + t_2) = \delta : t_1 + \delta : t_2$.
10. *If $\phi \vdash t = u$ then $\phi \vdash \delta : t = \delta : u$ for any δ.*
11. $\delta_1 : t + \delta_2 : t = (\delta_1 \vee \delta_2) : t$.
12. $\delta : \gamma \rightarrow t = (\delta \wedge \gamma) : \gamma \rightarrow t$.
13. $\delta : \gamma \rightarrow t = \gamma \rightarrow \delta : t$.
14. $\phi \vdash t = u \;\Rightarrow\; \phi \vdash \delta : t = \delta : u$.
15. $\mathit{ff} \rightarrow t = \mathbf{0}$.
16. $\mathbf{tt} : \mathbf{0} = \mathbf{0}$.
17. *From [LY02]:* $\phi \vdash a(\mathbf{x}).t = a(\mathbf{x}).\phi{\downarrow_{\mathbf{x}}}{\Uparrow} \rightarrow t$.
18. *From [LY02]: Suppose Ψ is a ϕ-partition and $\psi \vdash t = u$ for each $\psi \in \Psi$, then $\phi \vdash t = u$.*

Proof: (of Lemma 3.1.1)

$$\begin{aligned}
(\phi \Rightarrow \gamma) \;&\Rightarrow\; \phi \wedge \neg\gamma = \mathbf{ff} && (3.2)\\
&\Rightarrow\; \{\textsf{ABSURD and (3.2)}\}\\
&\quad\;\; \phi \wedge \neg\gamma \vdash \mathbf{0} = u. && (3.3)\\
&\Rightarrow\; \{\text{Applying } \textsf{PARTITION} \text{ and the hypothesis } \phi \vdash t = u\}\\
&\quad\;\; \phi \wedge \gamma \vdash t = u && (3.4)\\
&\Rightarrow\; \{\text{Applying } \textsf{GUARD} \text{ on (3.3), (3.4) }\}\\
&\quad\;\; \phi \vdash \gamma \rightarrow t = u.
\end{aligned}$$

□

Proof: (of Lemma 3.1.2)

$$\begin{aligned}
(\phi \Rightarrow \neg\delta) \;&\Rightarrow\; \phi \wedge \delta = \mathbf{ff} && (3.5)\\
&\Rightarrow\; \{\textsf{ABSURD and (3.5)}\}\\
&\quad\;\; \phi \wedge \delta \vdash \mathbf{tt} : t = u. && (3.6)
\end{aligned}$$

$\Rightarrow$ {Applying PARTITION and the hypothesis $\phi \vdash t = u$}

$$\phi \wedge \neg\delta \vdash t = u \tag{3.7}$$

$\Rightarrow$ {Applying DEADLINE on (3.6), (3.7) }

$\phi \vdash \delta : t = u$.

□

Proof: (of Lemma 3.1.3)

$$\phi \Rightarrow (\gamma \wedge \neg\delta) \text{ and } \phi \vdash t = u \tag{3.8}$$

$\Rightarrow$ $\phi \Rightarrow \gamma$ and $\phi \vdash t = u$

$\Rightarrow$ { By Lemma 3.1.1}

$\phi \Rightarrow \neg\delta$ and $\phi \vdash \gamma \rightarrow t = u$

$\Rightarrow$ { By Lemma3.1.2}

$\phi \vdash \delta : \gamma \rightarrow t = u$

$\wedge$ { Similarly by (3.8)}

$\Rightarrow$ $\phi \Rightarrow \neg\delta$ and $\phi \vdash t = u$

$\Rightarrow$ { By Lemma 3.1.2}

$\phi \Rightarrow \gamma$ and $\phi \vdash \delta : t = u$

$\Rightarrow$ { By Lemma 3.1.1}

$\phi \vdash \gamma \rightarrow \delta : t = u$

□

Proof: (of Lemma 3.1.4)

$\Rightarrow$ { by Lemma 3.1.1 and since $\phi \Rightarrow \phi$}

$\phi \vdash \phi \rightarrow t = t$

$\Rightarrow$ { By SUBSTITUTION, S2 and EQUIV}

$$\phi \vdash t = t + \phi \rightarrow t \tag{3.9}$$

$\wedge$ { Since $\neg\phi \wedge \phi = \mathbf{ff}$ and by ABSURD}

$$\neg\phi \wedge \phi \vdash t = \mathbf{0} \tag{3.10}$$

$\Rightarrow$ { By EQUIV}

$$\neg\phi \wedge \neg\phi \vdash \mathbf{0} = \mathbf{0} \tag{3.11}$$

$\Rightarrow$ { Applying GUARD on (3.10) and (3.11)}

$$\neg\phi \vdash \phi \rightarrow t = \mathbf{0} \tag{3.12}$$

$\Rightarrow$ { By SUBSTITUTION, S1 and EQUIV}

$$\phi \vdash t = t + \phi \rightarrow t \tag{3.13}$$

$\Rightarrow$ { applying PARTITION on (3.9) and (3.13), and EQUIV}

$t = t + \phi \rightarrow t$

□

Proof: (of Lemma 3.1.5)

$\Rightarrow$ { by Prop 4.1 [LY02]}
$\gamma_1 \rightarrow \gamma_2 \rightarrow t = (\gamma_1 \wedge \gamma_2) \rightarrow t$

□

Proof: (of Lemma 3.1.6)

$\Rightarrow$ { by (3.12)}
$\neg\gamma \vdash \gamma \rightarrow t_1 = \mathbf{0}$ and $\neg\gamma \vdash \gamma \rightarrow t_2 = \mathbf{0}$

$\Rightarrow$ { By S1 and EQUIV}
$\neg\gamma \vdash \mathbf{0} = \gamma \rightarrow t_1 + \mathbf{0}$ (3.14)

AND { By SUBSTITUTION}
$\neg\gamma \vdash \gamma \rightarrow t_1 + \mathbf{0} = \gamma \rightarrow t_1 + \gamma \rightarrow t_2$ (3.15)

$\Rightarrow$ { by EQUIV on (3.14) and (3.15)}
$\neg\gamma \vdash \mathbf{0} = \gamma \rightarrow t_1 + \gamma \rightarrow t_2$ (3.16)

AND {By Lemma 3.1.1}
$\gamma \vdash \gamma \rightarrow t_1 = t_1$ and $\gamma \vdash \gamma \rightarrow t_2 = t_2$

$\Rightarrow$ { By SUBSTITUTION}
$\gamma \vdash \gamma \rightarrow t_1 + t_2 = t_1 + t_2$ and $\gamma \vdash \gamma \rightarrow t_1 + \gamma \rightarrow t_2 = \gamma \rightarrow t_1 + t_2$

$\Rightarrow$ { By EQUIV}
$\gamma \vdash t_1 + t_2 = \gamma \rightarrow t_1 + \gamma \rightarrow t_2$ (3.17)

$\Rightarrow$ { Applying GUARD on (3.16) and (3.17)}
$\gamma \rightarrow (t_1 + t_2) = \gamma \rightarrow t_1 + \gamma \rightarrow t_2$ (3.18)

□

Proof: (of Lemma 3.1.7)

$\Rightarrow$ { by Lemma 3.1.4 and since $\phi \Rightarrow \phi$}
$\mathbf{tt} \vdash t = t + \gamma_2 \rightarrow t$

$\Rightarrow$ { since $\gamma_1 \Rightarrow \mathbf{tt}$, by PARTITION}
$\gamma_1 \vdash t = t + \gamma_2 \rightarrow t$ (3.19)

AND { By Lemma 3.1.1 and since $\gamma_1 \Rightarrow \gamma_1$}
$\gamma_1 \vdash t = \gamma_1 \rightarrow t$

$\Rightarrow$ { by SUBSTITUTION }
$\gamma_1 \vdash t + \gamma_2 \rightarrow t = \gamma_1 \rightarrow t + \gamma_2 \rightarrow t$ (3.20)

$\Rightarrow$ { by EQUIV on (3.19) and (3.20)}
$\gamma_1 \vdash t = \gamma_1 \rightarrow t + \gamma_2 \rightarrow t$ (3.21)

AND {Applying the same procedure on γ_2 }
$\gamma_2 \vdash t = \gamma_1 \rightarrow t + \gamma_2 \rightarrow t$ (3.22)

$\Rightarrow$ { By PARTITION on (3.21) and (3.22)}
$(\gamma_1 \vee \gamma_2) = \gamma_1 + t + \gamma_2 \rightarrow t$

□

Proof: (of Lemma 3.1.8)

$\Rightarrow$ { by EQUIV and PARTITION}
$\neg(\delta_1 \vee \delta_2) \vdash t = t$
$\Rightarrow$ { By Lemma 3.1.2 and logics}
$(\neg\delta_1 \wedge \neg\delta_2) \vdash t = (\delta_1 \vee \delta_2) : t$ (3.23)
AND { since $((\neg\delta_1 \wedge \delta_2) \wedge \neg(\delta_1 \vee \delta_2)) = \mathbf{ff}$ and by ABSURD}
$((\neg\delta_1 \wedge \delta_2) \wedge \neg(\delta_1 \vee \delta_2)) \vdash \mathbf{tt} : t = t$ (3.24)
AND { by EQUIV }
$((\neg\delta_1 \wedge \delta_2) \wedge \neg(\delta_1 \vee \delta_2)) \vdash \mathbf{tt} : t = \mathbf{tt} : t$ (3.25)
$\Rightarrow$ { Applying DEADLINE on (3.24) and (3.25)}
$\neg\delta_1 \wedge \delta_2 \vdash \mathbf{tt} : t = (\delta_1 \vee \delta_2) : t$ (3.26)
$\Rightarrow$ { Applying DEADLINE on (3.23) and (3.26)}
$\neg\delta_1 \vdash \delta_2 : t = (\delta_1 \vee \delta_2) : t$ (3.27)
AND {since $(\delta_1 \wedge \neg(\delta_1 \vee \delta_2)) = \mathbf{ff}$ then by ABSURD }
$(\delta_1 \wedge \neg(\delta_1 \vee \delta_2)) \vdash \mathbf{tt} : \delta_2 : t = t$ (3.28)
AND { by EQUIV and PARTITION}
$(\delta_1 \wedge \neg\delta_2) \vdash \mathbf{tt} : t = \mathbf{tt} : t$ (3.29)
AND {By D1, EQUIV and PARTITION }
$(\delta_1 \wedge \delta_2) \vdash \mathbf{tt} : \mathbf{tt} : t = \mathbf{tt} : t$ (3.30)
$\Rightarrow$ { Applying DEADLINE on (3.29) and (3.30)}
$\delta_1 \wedge (\delta_1 \vee \delta_2) \vdash \delta_2 : \mathbf{tt} : t = \mathbf{tt} : t$
$\Rightarrow$ { by D2}
$\delta_1 \wedge (\delta_1 \vee \delta_2) \vdash \mathbf{tt} : \delta_2 : t = \mathbf{tt} : t$ (3.31)
$\Rightarrow$ { Applying DEADLINE on (3.28) and (3.31) and EQUIV}
$\delta_1 \vdash \mathbf{tt} : \delta_2 : t = (\delta_1 \vee \delta_2) : t$ (3.32)
$\Rightarrow$ { Applying DEADLINE on (3.27) and (3.32) and EQUIV}
$\mathbf{tt} \vdash \delta_1 : \delta_2 : t = (\delta_1 \vee \delta_2) : t$

□

Proof: (of Lemma 3.1.9) First we prove the following small lemma

If $\delta \Rightarrow \phi$ then $\phi \vdash \delta : t = \mathbf{tt} : t$ (3.33)

$(\delta \Rightarrow \phi) \quad \Rightarrow \quad ((\phi \wedge \neg\delta) = \mathbf{ff})$

$\Rightarrow$ { By ABSURD}

$(\phi \wedge \neg\delta) \vdash t = \mathbf{tt} : t$ (3.34)

AND { By EQUIV}

$(\phi \wedge \delta) \vdash \mathbf{tt} : t = \mathbf{tt} : t$ (3.35)

$\Rightarrow$ { Applying DEADLINE on (3.34) and (3.35) }

$\phi \vdash \delta : t = \mathbf{tt} : t$

Now the proof of Lemma 3.1.9 follows

$\Rightarrow$ { by Lemma 3.1.2}

$\neg\delta \vdash t_1 = \delta : t_1$

$\Rightarrow$ { By SUBSTITUTION}

$\neg\delta \vdash t_1 + t_2 = \delta : t_1 + t_2$ (3.36)

AND { By Lemma 3.1.2 and SUBSTITUTION as above}

$\neg\delta \vdash \delta : t_1 + t_2 = \delta : t_1 + \delta : t_2$ (3.37)

$\Rightarrow$ { By EQUIV of(3.36) and (3.37) }

$\neg\delta \vdash t_1 + t_2 = \delta : t_1 + \delta : t_2$ (3.38)

AND { by (3.33) and EQUIV}

$\delta \vdash \mathbf{tt} : t_1 = \delta : t_1$

$\Rightarrow$ { Applying SUBSTITUTION}

$\delta \vdash \mathbf{tt} : t_1 + \mathbf{tt} : t_2 = \delta : t_1 + \mathbf{tt} : t_2$ (3.39)

$\Rightarrow$ { Again (3.33) and EQUIV}

$\delta \vdash \mathbf{tt} : t_2 = \delta : t_2$

$\Rightarrow$ { Applying SUBSTITUTION}

$\delta \vdash \delta : t_1 + \mathbf{tt} : t_2 = \delta : t_1 + \delta : t_2$ (3.40)

$\Rightarrow$ { by EQUIV of (3.39) and (3.40)}

$\delta \vdash \mathbf{tt} : t_1 + \mathbf{tt} : t_2 = \delta : t_1 + \delta : t_2$ (3.41)

$\Rightarrow$ {By D4 }

$\delta \vdash \mathbf{tt} : (t_1 + t_2) = \delta : t_1 + \delta : t_2$ (3.42)

$\Rightarrow$ { Applying DEADLINE on (3.38) and (3.42)}

$\mathbf{tt} : (t_1 + t_2) = \delta : t_1 + \delta : t_2$

□

Proof: (of Lemma 3.1.16)

$\Rightarrow$ { By Lemma 3.1.15}

$\mathbf{ff} \rightarrow t = \mathbf{0}$

$\Rightarrow$ { by SUBSTITUTION }

$\mathbf{tt} : \mathbf{ff} \rightarrow t = \mathbf{tt} : \mathbf{0}$

$\Rightarrow$ { by U3 and EQUIV}
$\mathbf{ff} \rightarrow \mathbf{tt} : t = \mathbf{tt} : \mathbf{0}$
$\Rightarrow$ { By Lemma 3.1.15 and EQUIV}
$\mathbf{0} = \mathbf{tt} : \mathbf{0}$

□

The proof of Lemmas 3.1.10 – 3.1.15 are straight forward applications of the above lemmas, and their proofs are omitted.

The following Lemma helps to gather summands that only differ in their guards and deadlines

Lemma 3.2 *Let δ_1, δ_2, γ_1, and γ_2 be predicates, then the equation*

$$\delta_1 : \gamma_1 \rightarrow t + \delta_2 : \gamma_2 \rightarrow t = \Big((\delta_1 \wedge \gamma_1) \vee (\delta_2 \wedge \gamma_2)\Big) : (\gamma_1 \vee \gamma_2) \rightarrow t$$

is provable. In particular if $\delta_1 \Rightarrow \gamma_1$ and $\delta_2 \Rightarrow \gamma_2$ then

$$\delta_1 : \gamma_1 \rightarrow t + \delta_2 : \gamma_2 \rightarrow t = (\delta_1 \vee \delta_2) : (\gamma_1 \vee \gamma_2) \rightarrow t$$

Proof: First we shall prove for the case when one of the deadlines (say δ_2) is false. That is we need to prove

$$\delta_1 : \gamma_1 \rightarrow t + \gamma_2 \rightarrow t \quad = \quad (\delta_1 \wedge \gamma_1) : (\gamma_1 \vee \gamma_2) \rightarrow t \tag{3.43}$$

by DEADLINE we need to prove that

$$\neg\delta_1 \vdash \delta_1 : \gamma_1 \rightarrow t + \gamma_2 \rightarrow t \quad = \quad (\gamma_1 \vee \gamma_2) \rightarrow t \tag{3.44}$$

$$\delta_1 \vdash \delta_1 : \gamma_1 \rightarrow t + \gamma_2 \rightarrow t \quad = \quad \mathbf{tt} : (\gamma_1 \vee \gamma_2) \rightarrow t \tag{3.45}$$

Using DEADLINE, ABSURD and EQUIV we can easily prove that

$$\neg\delta_1 \vdash \delta_1 : \gamma_1 \rightarrow t \quad = \quad \gamma_1 \rightarrow t$$

Next, we can add $\gamma_2 \rightarrow t$ on both sides using SUBSTITUTION. The right hand side equation will be equal to $(\gamma_1 \vee \gamma_2) \rightarrow t$ using Lemma 3.1.7, which proves equation (3.44).

In order to prove (3.45) we use GUARD, Lemma 3.1.12 and Lemma 3.1.13 to decompose the problem into the following equations.

$$(\delta_1 \wedge \gamma_1) \wedge (\gamma_1 \vee \gamma_2) \vdash \delta_1 : \gamma_1 \rightarrow t + \gamma_2 \rightarrow t \quad = \quad \mathbf{tt} : t \tag{3.46}$$

$$(\delta_1 \wedge \gamma_1) \wedge \neg(\gamma_1 \vee \gamma_2) \vdash \delta_1 : \gamma_1 \rightarrow t + \gamma_2 \rightarrow t \quad = \quad \mathbf{0} \tag{3.47}$$

$\Rightarrow$ {By DEADLINE, ABSURD and EQUIV}
$(\delta_1 \wedge \gamma_1) \vdash \delta_1 : t = \mathbf{tt} : t$
$\Rightarrow$ {Applying Lemma 3.1.1, Lemma 3.1.13 and EQUIV}
$(\delta_1 \wedge \gamma_1) \vdash \delta_1 \rightarrow \gamma_1 : t = \mathbf{tt} : t$

$$\begin{array}{rl}
\Rightarrow & \{\text{By SUBSTITUTION}\} \\
& (\delta_1 \wedge \gamma_1) \vdash \delta_1 \rightarrow \gamma_1 : t + \gamma_2 \rightarrow t = \mathbf{tt} : t + \gamma_2 \rightarrow t \\
\Rightarrow & \{\text{By Lemma 3.1.11 and EQUIV}\} \\
& (\delta_1 \wedge \gamma_1) \vdash \delta_1 \rightarrow \gamma_1 : t + \gamma_2 \rightarrow t = \mathbf{tt} : t \\
\Rightarrow & \{\text{By CONSEQUENCE since } (\delta_1 \wedge \gamma_1) = (\delta_1 \wedge \gamma_1) \wedge (\gamma_1 \wedge \gamma_2)\ \} \\
& (\delta_1 \wedge \gamma_1) \wedge (\gamma_1 \wedge \gamma_2) \vdash \delta_1 \rightarrow \gamma_1 : t + \gamma_2 \rightarrow t = \mathbf{tt} : t
\end{array}$$

This proves equation (3.46). Note that $(\delta_1 \wedge \gamma_1) \wedge \neg(\gamma_1 \vee \gamma_2) = \mathbf{ff}$ and by ABSURD and CONSEQUENCE we prove (3.47), which completes the proof of (3.45).

The same proof applies for the case when δ_1 is false. Finally we group these two cases and prove the present Lemma as follows.

$$\begin{array}{rl}
& (\delta_1 \wedge \gamma_1) : \gamma_1 \rightarrow t + (\delta_2 \wedge \gamma_2) : \gamma_2 \rightarrow t \\
= & \{\text{Applying (3.45) twice}\} \\
& (\delta_1 \wedge \gamma_1) : \gamma_1 \rightarrow t + \gamma_1 \rightarrow t + (\delta_2 \wedge \gamma_2) : \gamma_2 \rightarrow t + \gamma_2 \rightarrow t \\
= & \{\text{By S1-S4 and Lemma 3.1.6}\} \\
& (\delta_1 \wedge \gamma_1) : \gamma_1 \rightarrow t + (\gamma_1 \vee \gamma_2) \rightarrow t + (\delta_2 \wedge \gamma_2) : \gamma_2 \rightarrow t + (\gamma_1 \vee \gamma_2) \rightarrow t \\
= & \{\text{Applying (3.45) twice}\} \\
& (\delta_1 \wedge \gamma_1) : (\gamma_1 \vee \gamma_2) \rightarrow t + (\delta_2 \wedge \gamma_2) : (\gamma_1 \vee \gamma_2) \rightarrow t \\
= & \{\text{By Lemma Lemma 3.1.11}\} \\
& (\delta_1 \wedge \gamma_1) \vee (\delta_2 \wedge \gamma_2) : (\gamma_1 \vee \gamma_2) \rightarrow t
\end{array}$$

□

Lemma 3.3 is a generalization of the axiom DL in which deadlines of an action have the same impact on any number of summands as long as they are prefixed with the same action.

Lemma 3.3 *Let δ_i, γ_i be predicates, where i is a finite non-negative integer, then the following generalization equation of axiom* DL *is provable.*

$$\sum_{i=1}^{n} \Big(\delta_i : \gamma_i \rightarrow a(\mathbf{x}_i).t_i\Big) \quad = \quad \bigvee_{i=1}^{n} (\delta_i \wedge \gamma_i) : \left(\sum_{i=1}^{n} \gamma_i \rightarrow a(\mathbf{x}_i).t_i\right)$$

Proof: We will show for the case when $n = 3$. Using the same technique recursively, it is straightforward to show for arbitrary n.

$$\begin{array}{rl}
& \delta_1 : \gamma_1 \rightarrow a(\mathbf{x_1}).t_1 + \delta_2 : \gamma_2 \rightarrow a(\mathbf{x_2}).t_2 + \delta_3 : \gamma_3 \rightarrow a(\mathbf{x_3}).t_3 \\
= & \{\text{Applying S1-S4, Lemma 3.1.2, Lemma 3.1.11 and Lemma 3.1.12}\} \\
& (\delta_1 \wedge \gamma_1) : \gamma_1 \rightarrow a(\mathbf{x_1}).t_1 + \gamma_1 \rightarrow a(\mathbf{x_1}).t_1 + (\delta_1 \wedge \gamma_1) : \gamma_1 \rightarrow a(\mathbf{x_1}).t_1 + \gamma_1 \rightarrow a(\mathbf{x_1}).t_1 + \\
& (\delta_2 \wedge \gamma_2) : \gamma_2 \rightarrow a(\mathbf{x_2}).t_2 + \gamma_2 \rightarrow a(\mathbf{x_2}).t_2 + (\delta_2 \wedge \gamma_2) : \gamma_2 \rightarrow a(\mathbf{x_2}).t_2 + \gamma_2 \rightarrow a(\mathbf{x_2}).t_2 + \\
& (\delta_3 \wedge \gamma_3) : \gamma_3 \rightarrow a(\mathbf{x_2}).t_3 + \gamma_3 \rightarrow a(\mathbf{x_3}).t_3 + (\delta_3 \wedge \gamma_3) : \gamma_3 \rightarrow a(\mathbf{x_2}).t_3 + \gamma_3 \rightarrow a(\mathbf{x_2}).t_3 +
\end{array}$$

$= \{\text{Applying S1-S4}\}$

$(\delta_1 \wedge \gamma_1) : \gamma_1 \to a(\mathbf{x_1}).t_1 + \gamma_2 \to a(\mathbf{x_2}).t_2 + (\delta_1 \wedge \gamma_1) : \gamma_1 \to a(\mathbf{x_1}).t_1 + \gamma_3 \to a(\mathbf{x_3}).t_3 +$
$(\delta_2 \wedge \gamma_2) : \gamma_2 \to a(\mathbf{x_2}).t_2 + \gamma_1 \to a(\mathbf{x_1}).t_1 + (\delta_2 \wedge \gamma_2) : \gamma_2 \to a(\mathbf{x_2}).t_2 + \gamma_3 \to a(\mathbf{x_3}).t_3 +$
$(\delta_3 \wedge \gamma_3) : \gamma_3 \to a(\mathbf{x_3}).t_3 + \gamma_1 \to a(\mathbf{x_1}).t_1 + (\delta_3 \wedge \gamma_3) : \gamma_3 \to a(\mathbf{x_3}).t_3 + \gamma_2 \to a(\mathbf{x_2}).t_2$

$= \{\text{Applying DL six times}\}$

$(\delta_1 \wedge \gamma_1) : \big(\gamma_1 \to a(\mathbf{x_1}).t_1 + \gamma_2 \to a(\mathbf{x_2}).t_2\big) + (\delta_1 \wedge \gamma_1) : \big(\gamma_1 \to a(\mathbf{x_1}).t_1 + \gamma_3 \to a(\mathbf{x_3}).t_3\big) +$
$(\delta_2 \wedge \gamma_2) : \big(\gamma_2 \to a(\mathbf{x_2}).t_2 + \gamma_1 \to a(\mathbf{x_1}).t_1\big) + (\delta_2 \wedge \gamma_2) : \big(\gamma_2 \to a(\mathbf{x_2}).t_2 + \gamma_3 \to a(\mathbf{x_3}).t_3\big) +$
$(\delta_3 \wedge \gamma_3) : \big(\gamma_3 \to a(\mathbf{x_3}).t_3 + \gamma_1 \to a(\mathbf{x_1}).t_1\big) + (\delta_3 \wedge \gamma_3) : \big(\gamma_3 \to a(\mathbf{x_3}).t_3 + \gamma_2 \to a(\mathbf{x_2}).t_2\big)$

$= \{\text{Applying Lemma 3.1.9 and S1-S4 three times }\}$

$(\delta_1 \wedge \gamma_1) : \big(\gamma_1 \to a(\mathbf{x_1}).t_1 + \gamma_2 \to a(\mathbf{x_2}).t_2 + \gamma_3 \to a(\mathbf{x_3}).t_3\big) +$
$(\delta_2 \wedge \gamma_2) : \big(\gamma_1 \to a(\mathbf{x_1}).t_1 + \gamma_2 \to a(\mathbf{x_2}).t_2 + \gamma_3 \to a(\mathbf{x_3}).t_3\big) +$
$(\delta_3 \wedge \gamma_3) : \big(\gamma_1 \to a(\mathbf{x_1}).t_1 + \gamma_2 \to a(\mathbf{x_2}).t_2 + \gamma_3 \to a(\mathbf{x_3}).t_3\big) +$

$= \{\text{Applying Lemma 3.1.11 and S1-S4 twice }\}$

$$\bigvee_{i=1}^{3} (\delta_i \wedge \gamma_i) : \left(\sum_{i=1}^{3} \gamma_i \to a(\mathbf{x}_i).t_i \right)$$

□

3.5 Soundness

In the previous section, we provided axioms and inference rules to simplify and manipulate terms in $\mathbb{A}$. In this section, we prove the soundness of these inference rules with respect to ∇-bisimulation. Formally, the soundness of the proof system can be stated as follows

Theorem 3.1 *If $\phi \vdash t{=}u$ and ϕ is $\Uparrow$-closed then $(t, D)\rho \sim^{\nabla} (u, D)\rho$ for any $\rho{\models}\phi$ and $D{\subseteq}\mathcal{A}$.*

The usual way to prove soundness is to show that if $\phi \vdash t = u$ and ϕ is $\Uparrow$-closed then $t \sim^{\phi} u$. However as it is already noticed in [LY02], this approach will not work specially for GUARD and DEADLINE. For example, in order to derive $\phi \vdash \gamma \to t = u$, we need to show $\phi \wedge \gamma \vdash t = u$ and $\phi \wedge \neg\gamma \vdash \mathbf{0} = u$. Note that even if ϕ is $\Uparrow$-closed, $\phi \wedge \gamma$ may not be $\Uparrow$-closed. For this reason, we will first define an intermediate bisimulation relation, called *bisimulation up to d* denoted as $\sim^{\nabla}_{d}$. We start by defining $\sim^{\nabla}_{d}$ formally.

Definition 3.1 (∇-bisimulation up to *d*) *Two states p and q are* ∇-bisimilar up to *$\hat{d}$ for $\hat{d} \in \mathbb{R}_{\geq 0}$, notation $p \sim^{\nabla}_{\hat{d}} q$, if there is a family of symmetric relations $R_d \subseteq \mathcal{S} \times \mathcal{S}$, $0 \leq d \leq \hat{d}$ such that*

1. *$\forall d'{\in}\mathbb{R}_{\geq 0}, d'{<}d$, if $(p, q){\in}R_d$ and $p \xrightarrow{d'} p'$ then $\exists q' : q \xrightarrow{d'} q'$ and $(p', q') \in R_{d-d'}$.*

2. *$\forall l{\in}\{\Delta\}{\cup}\mathcal{A}_{\nabla}$, if $(p,q) \in R_d$ and $p \xrightarrow{l} p'$ then $\exists q' : q \xrightarrow{l} q'$ and $(p',q') \in R_d$.*

3. *$\forall a{\in}\mathcal{A}$, if $(p,q) \in R_d$ and $p \xrightarrow{a} p'$ then $\exists q' : q \xrightarrow{a} q'$ and $p' \sim^{\nabla} q'$.*

Lemma 3.4 1. *If $p \sim^{\nabla}_{d} q$ for all $d \in \mathbb{R}_{\geq 0}$ then $p \sim^{\nabla} q$.*

2. *Let ρ_i and d_i, $0 \leq i \leq n$, be s.t. $\rho_{i+1} = \rho_i + d_i$, $0 \leq i < n$. If $(t,D)\rho_i \sim^{\nabla}_{d_i} (u,D)\rho_i$ for all i such that $0 \leq i \leq n$, then $(t,D)\rho_0 \sim^{\nabla}_{d} (u,D)\rho_0$ where $d = d_0{+}\cdots{+}d_n$.*

3. *$(t,D)\rho{\sim}^{\nabla}_{\hat{d}}(u,D)\rho$ implies $(t,D)(\rho{+}\hat{d}'){\sim}^{\nabla}_{\hat{d}''}(t,D)(\rho{+}\hat{d}')$ for any $\hat{d}'{\leq}\hat{d}$ and $\hat{d}'' \leq \hat{d}{-}\hat{d}'$.*

4. *$\sim^{\nabla}_{d}$ is transitive.*

Proof: Proofs of items 1 and 2 proceed as [LY02, Lemma 4.10]. Proof of item 4 follows standard arguments. For item 3, suppose $\{R_d\}_{d \leq \hat{d}}$ witnesses $(t,D)\rho \sim^{\nabla}_{\hat{d}} (u,D)\rho$. First notice that for $((t,D)\rho,(u,D)\rho) \in R_{\hat{d}}$, $(t,D)\rho \xrightarrow{\nabla_{\mathcal{A}}} (t,\mathcal{A})\rho \xrightarrow{\hat{d}'} (t,\mathcal{A})(\rho{+}\hat{d}') \xrightarrow{\Delta} (t,\varnothing)(\rho{+}\hat{d}') \xrightarrow{\nabla_D} (t,D)(\rho{+}\hat{d}')$, (by DROP, DELAY, and UNDROP in Fig. 3.2) implies, by Def. 3.1, that $(u,D)\rho \xrightarrow{\nabla_{\mathcal{A}}} (u,\mathcal{A})\rho \xrightarrow{\hat{d}'} (u,\mathcal{A})(\rho{+}\hat{d}') \xrightarrow{\Delta} (u,\varnothing)(\rho{+}\hat{d}') \xrightarrow{\nabla_D} (u,D)(\rho{+}\hat{d}')$ and $((t,D)(\rho{+}\hat{d}'),(u,D)(\rho{+}\hat{d}')) \in R_{\hat{d}-\hat{d}'}$. It is now straightforward to show that $\{R_{(\hat{d}-\hat{d}')+d}\}_{d \leq \hat{d}''}$ witnesses $t(\rho+\hat{d}') \sim^{\nabla}_{\hat{d}''} u(\rho{+}\hat{d}')$. □

In the following lemmas we state some properties of deadlines and guards, which will be used later to prove soundness.

Lemma 3.5 *For $\hat{d}, d \in \mathbb{R}_{\geq 0}$ and $D \subseteq \mathcal{A}$*

1. *$(\delta : t,D)\rho \sim^{\nabla}_{\hat{d}} (t,D)\rho$ if for all $d < \hat{d} : \rho + d \models \neg\delta$*

2. *$(\delta : t,D)\rho \sim^{\nabla}_{\hat{d}} (\boldsymbol{tt} : t,D)\rho$ if for all $d < \hat{d} : \rho + d \models \delta$*

3. *$(t,D)\rho \sim^{\nabla}_{\hat{d}} (u,D)\rho$ implies $(\boldsymbol{tt} : t,D)\rho \sim^{\nabla}_{\hat{d}} (\boldsymbol{tt} : u,D)\rho$*

4. *$(\gamma \rightarrow t,D)\rho \sim^{\nabla}_{\hat{d}} (t,D)\rho$ if for all $d \leq \hat{d} : \rho + d \models \gamma$*

5. *$(\gamma \rightarrow t,D)\rho \sim^{\nabla}_{\hat{d}} (\mathbf{0},D)\rho$ if for all $d \leq \hat{d} : \rho + d \models \neg\gamma$*

Proof: It is routine to prove that families $\{R_d \cup R_d^{-1}\}_{0 \leq d \leq \hat{d}}$, respectively defined in the following, satisfy conditions of Def. 3.1.

1. $R_d = \{((\delta : t,D)\rho,(t,D)\rho) \mid \forall d' < d : \rho + d \models \neg\delta\}$

2. $R_d = \{((\delta : t,D)\rho,(\mathbf{tt} : t,D)\rho) \mid \forall d' < d : \rho + d \models \delta\}$

3. $R_d = \{((\mathbf{tt} : t, D)\rho, (\mathbf{tt} : u, D)\rho) \mid (t, D)\rho \sim^{\nabla}_{d} (u, D)\rho\}$
4. $R_d = \{((\gamma \to t, D)\rho, (t, D)\rho) \mid \forall d' \leq d : \rho + d' \models \gamma\}$
5. $R_d = \{((\gamma \to t, D)\rho, (\mathbf{0}, D)\rho) \mid \forall d' \leq d : \rho + d' \models \neg\gamma\}$

□

Soundness rests on the following lemmas.

Lemma 3.6 *If $\phi \vdash t = u$ then $(t, D)\rho \sim^{\nabla}_{\hat{d}} (u, D)\rho$ for all $D \subseteq \mathcal{A}$, ρ, and $\hat{d} \in \mathbb{R}_{\geq 0}$ such that $\forall d \leq \hat{d} : \rho + d \models \phi$.*

Proof: The proof proceeds by induction on the depth of the proof tree. The base case corresponds to all axioms. That is, for every axiom $\phi \vdash t = u$ find a family $\{R_d\}_{d \leq \hat{d}}$ witnessing $(t, D)\rho \sim^{\nabla}_{\hat{d}} (u, D)\rho$. This is routine and we omit it.

For the induction step, we consider the inference rules separately. For each rule, we assume that the lemma holds in its premises and prove that it also holds in its conclusion. We only show a few representative cases. In particular, soundness of UFI is proved in Lemma 3.7.

ACTION: By induction $(t, D)\rho \sim^{\nabla}_{\hat{d}} (u, D)\rho$, for any D, ρ, $\hat{d}$, s.t. $\forall d \leq \hat{d}$: $(\rho{+}d) \models \phi{\downarrow_{\mathbf{x}}}{\Uparrow}$. Since $\phi{\downarrow_{\mathbf{x}}}{\Uparrow}$ is $\Uparrow$-closed, by Lemma 3.4.1, $(t, D)\rho \sim^{\nabla} (u, D)\rho$. We show that $\{R_d\}_{d \leq \hat{d}}$ witnesses $(a(\mathbf{x}).t, D)\rho \sim^{\nabla}_{\hat{d}} (a(\mathbf{x}).u, D)\rho$ for all ρ s.t. $\forall d \leq \hat{d} : (\rho{+}d) \models \phi$, where

$$\begin{aligned} R_d = \{((a(\mathbf{x}).t, D)\rho, (a(\mathbf{x}).u, D)\rho) \mid D \subseteq \mathcal{A} \wedge \forall d' \leq \hat{d} - d : \rho{+}d' \models \phi \wedge \\ \forall d' \leq d : (t, D)(\rho{+}d')\{\mathbf{x}{:=}0\} \sim^{\nabla} (u, D)(\rho{+}d')\{\mathbf{x}{:=}0\}\}. \end{aligned}$$

Assume $((a(\mathbf{x}).t, D)\rho, (a(\mathbf{x}).u, D)\rho) \in R_d$. We do case analysis on all the four possible type of transitions.

delay transition: By rule DELAY (Fig. 3.2), we have (for any $d'' \leq d$) that

$$(a(\mathbf{x}).t, D)\rho \xrightarrow{d''} (a(\mathbf{x}).t, D)(\rho{+}d'') \text{ and}$$
$$(a(\mathbf{x}).u, D)\rho \xrightarrow{d''} (a(\mathbf{x}).u, D)(\rho{+}d'')$$

It remains to show that

$$((a(\mathbf{x}).t, D)(\rho{+}d''), (a(\mathbf{x}).u, D)(\rho{+}d'')) \in R_{d-d''}$$

Since $\forall d' \leq \hat{d} - d : \rho{+}d' \models \phi$ holds by assumption, $\forall d' \leq \hat{d} - d$: $(\rho{+}d')\{\mathbf{x}{:=}0\} \models \phi{\downarrow_{\mathbf{x}}}{\Uparrow}$ also hods by Def. of${\downarrow_{\mathbf{x}}}$ and ${\Uparrow}$. By induction hypothesis and observation above, $\forall d' \leq \hat{d} - d : (t, D)(\rho{+}d')\{\mathbf{x}{:=}0\} \sim^{\nabla}$ $(u, D)(\rho{+}d')\{\mathbf{x}{:=}0\}$. In particular,

$$\forall d''' \leq \hat{d} - (d - d'') : \ (\rho{+}d''{+}d''')\{\mathbf{x}{:=}0\} \models \phi{\downarrow_{\mathbf{x}}}{\Uparrow}$$

$\wedge$
$$(t, D)(\rho{+}d''{+}d''')\{\mathbf{x}{:=}0\} \sim^{\nabla} (u, D)(\rho{+}d''{+}d''')\{\mathbf{x}{:=}0\}$$

By Def. of $R_{d-d''}$, we finally have that

$$((a(\mathbf{x}).t, D)(\rho{+}d''), (a(\mathbf{x}).u, D)(\rho{+}d'')) \in R_{d-d''}$$

drop and undrop transition: Let $l \in \{\Delta\} \cup \mathcal{A}_{\nabla}$. Then, by DROP or UNDROP (Fig. 3.2), $(a(\mathbf{x}).t, D)\rho \xrightarrow{l} (a(\mathbf{x}).t, E)\rho$ and $(a(\mathbf{x}).u, D)\rho \xrightarrow{l} (a(\mathbf{x}).u, E)\rho$ for any $E \subseteq \mathcal{A}$. Besides, $((a(\mathbf{x}).t, E)\rho, (a(\mathbf{x}).u, E)\rho) \in R_d$, since for all $d' \leq \hat{d} - d$, $(\rho{+}d') \models \phi$ implies $(\rho{+}d')\{\mathbf{x}{:=}0\} \models \phi{\downarrow_{\mathbf{x}}}{\Uparrow}$ and by induction

$$(t, E)(\rho{+}d')\{\mathbf{x}{:=}0\} \sim^{\nabla} (u, D)(\rho{+}d')\{\mathbf{x}{:=}0\}$$

discrete transition: By ACTION (Fig.3.2), $(a(\mathbf{x}).t, D)\rho \xrightarrow{a} (t, \varnothing)\rho\{\mathbf{x}{:=}0\}$ and $(a(\mathbf{x}).u, D)\rho \xrightarrow{a} (u, \varnothing)\rho\{\mathbf{x}{:=}0\}$. Moreover, since $\rho \models \phi$ by assumption, $\rho\{\mathbf{x}{:=}0\} \models \phi{\downarrow_{\mathbf{x}}}{\Uparrow}$, and hence $(t, \varnothing)\rho\{\mathbf{x}{:=}0\} \sim^{\nabla} (u, D)\rho\{\mathbf{x}{:=}0\}$ by induction.

DEADLINE: We need to prove that $(\delta : t, D)\rho \sim^{\nabla}_{d} (u, D)\rho$ for all ρ s.t. $\forall d \leq \hat{d}$: $(\rho + d) \models \phi$. The interval $[\rho, \rho + \hat{d})$ can be divided by regions into finitely many subintervals $[\rho_0, \rho_1)$, $[\rho_1, \rho_1]$, (ρ_1, ρ_2), ..., $[\rho_n, \rho_n]$, (ρ_n, ρ_{n+1}), where $\rho_0 = \rho$ and $\rho_{i+1} = \rho_i + d_i$ for some $d_0, \ldots, d_n$ s.t. $\sum_{i=0}^{n} d_i = d$ in a way that each point $[\rho_i, \rho_i]$, or interval $(\rho_i, \rho_i{+}d_i)$ is entirely contained in a region (so they are entirely contained in $\phi \wedge \neg\delta$ or in $\phi \wedge \delta$). By Lemma 3.4.2, it is enough to prove $(\delta : t, D)\rho_i \sim^{\nabla}_{d_i} (u, D)\rho_i$ for all $1 \leq i \leq n$. We only consider the case of intervals $(\rho_i, \rho_i{+}d_i)$, the others follow in a similar manner.
Case $(\rho_i, \rho_i{+}d_i) \models \phi \wedge \neg\delta$. By Lemma 3.5.1 $(\delta : t, D)\rho_i \sim^{\nabla}_{d_i} (t, D)\rho_i$. Besides, by induction and Lemma 3.4.3. $(t, D)\rho_i \sim^{\nabla}_{d_i} (u, D)\rho_i$. Hence, by transitivity of $\sim^{\nabla}_{d_i}$, $(\delta : t, D)\rho_i \sim^{\nabla}_{d_i} (u, D)\rho_i$.
Case $(\rho_i, \rho_i{+}d_i) \models \phi \wedge \delta$. By Lemma 3.5.2, $(\delta : t, D)\rho_i \sim^{\nabla}_{d_i} (\mathbf{tt} : t, D)\rho_i$ By induction and Lemma 3.4.3, $(\mathbf{tt} : t, D)\rho_i \sim^{\nabla}_{d_i} (u, D)\rho_i$. Therefore, by transitivity of $\sim^{\nabla}_{d_i}$, we have: $(\delta : t, D)\rho_i \sim^{\nabla}_{d_i} (u, D)\rho_i$.

GUARD: Using similar argument as in DEADLINE, we only need to prove that $(\gamma{\rightarrow}t, D)\rho_i \sim^{\nabla}_{d_i} (u, D)\rho_i$ for all $1 \leq i \leq n$.

Case: $(\rho_i, \rho_i{+}d_i) \models \phi \wedge \gamma$

$\Rightarrow$ {By Lemma 3.5.4}
$(\gamma{\rightarrow}t, D)\rho_i \sim^{\nabla}_{d_i} (t, D)\rho_i$

$\Rightarrow$ {By transitivity of $\sim^{\nabla}_{d_i}$ since $(t, D)\rho_i \sim^{\nabla}_{d_i} (u, D)\rho_i$, by induction and Lemma 3.4.3.}

$(\gamma{\to}t, D)\rho_i \sim^{\nabla}_{d_i} (u, D)\rho_i$

Case: $(\rho_i, \rho_i{+}d_i) \models \phi \wedge \neg\gamma$

$\Rightarrow$ {By Lemma 3.5.5}
$(\gamma{\to}t, D)\rho_i \sim^{\nabla}_{d_i} (\mathbf{0}, D)\rho_i$

$\Rightarrow$ {By induction and Lemma 3.4.3, $(\mathbf{0}, D)\rho_i \sim^{\nabla}_{d_i} (u, D)\rho_i$. Then, by transitivity of $\sim^{\nabla}_{d_i}$, we have:}
$(\gamma{\to}t, D)\rho_i \sim^{\nabla}_{d_i} (u, D)\rho_i$

SUBSTITUTION on choice: Suppose $(t, D)\rho \sim^{\nabla}_{\hat{d}} (u, D)\rho$, and suppose $\{R_d\}_{0\leq d\leq \hat{d}}$ witnesses it. We show that $\{R'_d\}_{0\leq d\leq \hat{d}}$ with $R'_d = \{((t + s, D)\rho, (u + s, D)\rho) \mid ((t, D)\rho, (u, D)\rho) \in R_d\}$, witnesses $(t + s, D)\rho \sim^{\nabla}_{\hat{d}} (u + s, D)\rho$. For all $d \leq \hat{d}$, suppose $((t + s, D)\rho, (u + s, D)\rho) \in R'_d$. We show the case of delay transition, the other cases are easier. For $d' < d$ we calculate:

$(t + s, D)\rho \xrightarrow{d'} (t + s, D)(\rho{+}d')$
$\Leftrightarrow$ {By definition of DELAY}
$\forall d'' < d' : (\rho{+}d'') \models \neg dl(t + s, D)$
$\Leftrightarrow$ {By definition of dl}
$\forall d'' < d' : (\rho{+}d'') \models \neg(dl(t, D) \vee dl(s, D))$
$\Leftrightarrow$ {Logic}
$\forall d'' < d' : (\rho{+}d'') \models \neg dl(t, D)$ and $\forall d'' < d' : (\rho{+}d'') \models \neg dl(s, D)$
$\Leftrightarrow$ {By definition of DELAY}
$(t, D)\rho \xrightarrow{d'} (t, D)(\rho{+}d')$ and $\forall d'' < d' : (\rho{+}d'') \models \neg dl(s, D)$
$\Rightarrow$ {Since $((t, D)\rho, (u, D)\rho) \in R_d$}
$(u, D)\rho \xrightarrow{d'} (u, D)(\rho{+}d')$, $((t, D)(\rho{+}d'), (u, D)(\rho{+}d')) \in R_{d-d'}$ and $\forall d'' < d' : (\rho{+}d'') \models \neg dl(s, D)$
$\Leftrightarrow$ {By definition of DELAY and $R'_{d-d'}$}
$\forall d'' < d' : (\rho{+}d'') \models \neg dl(t, D)$, $\forall d'' < d' : (\rho{+}d'') \models \neg dl(s, D)$ and $((t + s, D)(\rho{+}d'), (u + s, D)(\rho{+}d')) \in R'_{d-d'}$
$\Leftrightarrow$ {Logic}
$\forall d'' < d' : (\rho{+}d'') \models \neg(dl(u, D) \vee dl(s, D))$ and
$((t + s, D)(\rho{+}d'), (u + s, D)(\rho{+}d')) \in R'_{d-d'}$
$\Leftrightarrow$ {By definition of dl and DELAY}
$(u + s, D)\rho \xrightarrow{d'} (u + s, D)(\rho{+}d')$ and

$$((t+s,D)(\rho{+}d'),(u+s,D)(\rho{+}d')) \in R'_{d-d'}$$

SUBSTITUTION on urgency: First of all notice that if $(t,D)\rho \sim^{\nabla}_{\hat{d}} (u,D)\rho$ then $\rho \models gd(t,D)$ iff $\rho \models gd(u,D)$. From Lemma 3.4.3 it follows that $(\rho{+}d) \models gd(t,D)$ iff $(\rho{+}d) \models gd(u,D)$ for all $d < d'$. Call this observation $(\star)$.

From this observation notice that $(\rho{+}d) \models gd(t, I(t)-I(u)) \Leftrightarrow \mathbf{ff}$ for all $d < d'$ and hence $(\rho{+}d) \models \neg dl(\mathbf{tt} : t, I(t)-I(u)) \Leftrightarrow dl(\mathbf{tt} : u, I(t)-I(u))$. Symmetrically, it holds for t and u exchanged. So, w.l.o.g., we will suppose that $I(t) = I(u)$.

Now, we proceed in a similar fashion as the previous case. Define $R'_d = \{((\mathbf{tt} : t,D)\rho,(\mathbf{tt} : u,D)\rho) \mid ((t,D)\rho,(u,D)\rho) \in R_d\}$ provided $\{R_d\}_{0\leq d\leq\hat{d}}$ witnesses $(t,D)\rho \sim^{\nabla}_{\hat{d}} (u,D)\rho$. For all $d \leq \hat{d}$, suppose $((\mathbf{tt} : t,D)\rho,(\mathbf{tt} : u,D)\rho) \in R'_d$. We show the case of delay transition, the other cases are easier, and in this case we only consider $I(t)\cap D \neq \varnothing$ (and hence $I(u)\cap D \neq \varnothing$) since the case $I(t)\cap D = \varnothing$ is simpler. For $d' < d$ we calculate:

$$\begin{array}{ll}
 & (\mathbf{tt} : t,D)\rho \xrightarrow{d'} (\mathbf{tt} : t,D)(\rho{+}d') \\
\Leftrightarrow & \{\text{By definition of DELAY}\} \\
 & \forall d'' < d' : (\rho{+}d'') \models \neg dl(\mathbf{tt} : t,D) \\
\Leftrightarrow & \{\text{By definition of } dl\} \\
 & \forall d'' < d' : (\rho{+}d'') \models \neg((\mathbf{tt} \wedge gd(t,D)) \vee dl(t,D)) \\
\Leftrightarrow & \{\text{Logic}\} \\
 & \forall d'' < d' : (\rho{+}d'') \models \neg gd(t,D) \quad \text{and} \quad \forall d'' < d' : (\rho{+}d'') \models \neg dl(t,D) \\
\Leftrightarrow & \{\text{By definition of DELAY}\} \\
 & \forall d'' < d' : (\rho{+}d'') \models \neg gd(t,D) \quad \text{and} \quad (t,D)\rho \xrightarrow{d'} (t,D)(\rho{+}d') \\
\Rightarrow & \{\text{Since } ((t,D)\rho,(u,D)\rho) \in R_d \text{ and by observation } (\star)\} \\
 & \forall d'' < d' : (\rho{+}d'') \models \neg gd(u,D), \\
 & (u,D)\rho \xrightarrow{d'} (u,D)(\rho{+}d'), \quad \text{and} \quad ((t,D)(\rho{+}d'),(u,D)(\rho{+}d')) \in R_{d-d'} \\
\Leftrightarrow & \{\text{Following the inverse reasoning and by definition of } R'_{d-d'}\} \\
 & (\mathbf{tt} : u,D)\rho \xrightarrow{d'} (\mathbf{tt} : u,D)(\rho{+}d') \quad \text{and} \\
 & ((\mathbf{tt} : t,D)(\rho{+}d'),(\mathbf{tt} : u,D)(\rho{+}d')) \in R_{d-d'}
\end{array}$$

□

The next lemma states soundness of UFI which amounts to proving that every set of equation has a unique solution.

Lemma 3.7 *Let terms v_i ($i{\in}I$) contain at most variables X_i ($i{\in}I$) which occur only guarded. Then, if*

1. $(t_j,D)\rho \sim^{\nabla} (v_j[t_i/X_i \mid i{\in}I],D)\rho$ *and*

2. $(u_j, D)\rho \sim^{\nabla} (v_j[u_i/X_i \mid i{\in}I], D)\rho$

then

$$(t_j, D)\rho \quad \sim^{\nabla} \quad (u_j, D)\rho$$

for all $j \in I$, $D \subseteq \mathcal{A}$, and valuation ρ.

Proof: We show that

$$R = \{((v[\tilde{t}/\tilde{X}], D)\rho, (v[\tilde{u}/\tilde{X}], D)\rho) \mid \mathit{Vars}(v) \subseteq \{X_i \mid i{\in}I\}\}$$

is a timed bisimulation up to $\sim^{\nabla}$ (we let $[\tilde{s}/\tilde{X}]$ denote $[s_i/X_i \mid i{\in}I]$). First notice that R is symmetric. The proof of the transfer property proceeds by case analysis on the type of the transition. Cases ∇_A and Δ are straightforward. Case $a \in \mathcal{A}$ follows by induction on the proof tree doing case analysis on the form of v like Proposition 14, Sec. 4.5 [Mil89a]. For the delay transition, we first consider the case in which $v \equiv X_j$.

Suppose that $(X_j[\tilde{t}/\tilde{X}], D)\rho \xrightarrow{d} (X_j[\tilde{t}/\tilde{X}], D)(\rho + d)$. Notice that $X_j[\tilde{t}/\tilde{X}] \equiv t_j$ and $(t_j, D)\rho \sim^{\nabla} (v_j[\tilde{t}/\tilde{X}], D)\rho$. Hence $(v_j[\tilde{t}/\tilde{X}], D)\rho \xrightarrow{d} (v_j[\tilde{t}/\tilde{X}], D)(\rho + d)$ and $(X_j[\tilde{t}/\tilde{X}], D)(\rho + d) \sim^{\nabla} (v_j[\tilde{t}/\tilde{X}], D)(\rho + d)$ (†). By DELAY, $\forall d' < d \quad \rho + d' \models \neg dl(v_j[\tilde{t}/\tilde{X}], \mathcal{A} - D)$. Since all X_i are guarded in v_j, $dl(v_j[\tilde{t}/\tilde{X}], \mathcal{A} - D) = dl(v_j[\tilde{u}/\tilde{X}], \mathcal{A} - D)$ and from here $(v_j[\tilde{u}/\tilde{X}], D)\rho \xrightarrow{d} (v_j[\tilde{u}/\tilde{X}], D)(\rho + d)$. Noticing that $v_j[\tilde{u}/\tilde{X}] \equiv u_j \equiv X_j[\tilde{u}/\tilde{X}]$, we have that $(X_j[\tilde{u}/\tilde{X}], D)\rho \xrightarrow{d} (X_j[\tilde{u}/\tilde{X}], D)(\rho + d)$ and $(v_j[\tilde{u}/\tilde{X}], D)(\rho + d) \sim^{\nabla} (X_j[\tilde{u}/\tilde{X}], D)(\rho + d)$ (‡). Using (†) and (‡), conclude that $(X_j[\tilde{t}/\tilde{X}], D)(\rho + d) \sim^{\nabla}{\circ}R{\circ}{\sim^{\nabla}} (X_j[\tilde{u}/\tilde{X}], D)(\rho + d)$. From here and Theorem 2.1, $dl(t_j, D) \Leftrightarrow dl(u_j, D)$ for any $j \in I$ and $D \subseteq \mathcal{A}$. Using this fact and induction on the structure of v, it is routine to prove that $dl(v[\tilde{t}/\tilde{X}], D) \Leftrightarrow dl(v[\tilde{u}/\tilde{X}], D)$. Using this equivalence, the proof of the transfer property on the delay transition for an arbitrary v is straightforward.

□

3.6 Completeness

In this section, we present the completeness theorem of the proof system, that is, whenever $t \sim^{\phi} u$ then $\phi \vdash t = u$. The proof of the theorem follows the arguments used by Milner [Mil84, Mil89b]. That is, we will first show that any term t can provably satisfy a special kind of equation E, called the *standard equation* (Lemma 3.8). Next we prove that if $t \sim^{\phi} u$ then both t and u provably satisfy a common standard equation E (Lemma 3.9). Finally from these two results we shall conclude $\phi \vdash t = u$ (Theorem 3.2).

3.6.1 Transforming Sets of Equations

First, we formally define equations, standard equations and what it means for a term to provably $\tilde{\phi}$-satisfy an equation.

Definition 3.2 *An* equation *set*

$$E : \{X_i = u_i \mid i \in I\}$$

is a finite non-empty indexed set of declarations, where the X_i's are pairwise distinct process variables, and the u_i's are terms.

Definition 3.3 *Given a vector of conditions $\tilde{\phi} = \{\phi_i \mid i{\in}I\}$ and a vector of terms $\tilde{t} = \{t_i \mid i{\in}I\}$, we say that*

$$\tilde{t} \textit{ provably } \tilde{\phi}\textit{-satisfies a set of equations } E : \{X_i = u_i \mid i{\in}I\}$$

iff, for all $i{\in}I$,

$$\phi_i \vdash t_i = u_i[\tilde{\phi}{\to}\tilde{t}/\tilde{X}]$$

Alternatively, we say that t provably ϕ-satisfies E, to mean that $\tilde{t}$ provably $\tilde{\phi}$-satisfies E when $\phi = \phi_1$ and $t = t_1$.

Definition 3.4 *An equation set E is* standard *iff each equation of E is of the form:*

$$X_i \ = \sum_{a \in \mathcal{A}} \delta_{ia} : \sum_{k \in K_{ia}} \gamma_{ika} \to a(\mathbf{x}_{ika}).X_{f(i,k,a)} + \sum_{W \in \mathbf{V}} \delta_{iW} : \gamma_{iW} \to W \qquad (3.48)$$

where the vector X_i is disjoint from the set $\mathbf{V}$, for all $a \in \mathcal{A}$, $\delta_{ia} \Rightarrow \bigvee_{k \in K_{ia}} \gamma_{ika}$, and for all $W \in \mathbf{V}$, $\delta_{iW} \Rightarrow \gamma_{iW}$. We call X_i the formal *variables of E, and $W \in \mathbf{V}$ the* free *variables of E. The set of equation E is called* closed *if $\mathbf{V} = \varnothing$.*

For example, $\{X_1 = (x_1{\geq}1) : (x_1{\geq}1) \to a_1(x_2).X_2 + \mathbf{ff} : \mathbf{tt} \to W, X_2 = (2{\leq}x_2{<}3) : ((x_2{<}x_1) \to a_2(x_1).X_1 + (x_2{\geq}x_1) \to a_2(x_2).X_2)\}$ is a standard set of equation, with formal variable set $\mathbf{X} = \{X_1, X_2\}$ and free variable set $\mathbf{V} = \{W\}$.

Lemma 3.8 *For any guarded term t with free variables $\mathbf{V}$ there exists a set of standard equations E, with free process variables in $\mathbf{V}$, which is provably **tt**-satisfied by t. In particular, if t is closed so is E.*

Proof: Like in [Mil89b], we proceed by induction on the structure of t. We only report the most relevant cases

Case $t \equiv \mathbf{0}$: It is easy to check that E containing the only equation

$$X = \sum_{a \in \mathcal{A}} \mathbf{ff} : \mathbf{ff} \to a(\varnothing).X + \sum_{W \in \mathbf{V}} \mathbf{ff} : \mathbf{ff} \to W$$

is satisfied by $\mathbf{0}$ (recall Lemma 3.1.15).

Case $t \equiv X,\ X \in \mathbf{V}$: Again by Lemma 3.1.15 the equation

$$E : \left\{ X_1 = \sum_{a \in \mathcal{A}} \mathbf{ff} : \mathbf{ff} \to a(\varnothing).X_1 + \sum_{W \in \mathbf{V} - \{X\}} \mathbf{ff} : \mathbf{ff} \to W \ + \ \mathbf{ff} : \mathbf{tt} \to X \right\}$$

is satisfied by X.

Case $t \equiv t' + t''$: By induction, t' and t'' satisfy sets of standard equations. Let them be E' and E'', respectively. Define the set of equations E containing all equations in E' and E'' and the new equation

$$\begin{aligned} X_1 = \sum_{a \in \mathcal{A}} (\delta'_{1a} \vee \delta''_{1a}) : \Bigg(& \sum_{k' \in K'_{1a}} \gamma'_{1k'a} \to a(\mathbf{x}_{1k'a}).X'_{f'(1,k',a)} \\ & + \sum_{k'' \in K''_{1a}} \gamma''_{1k''a} \to a(\mathbf{x}_{1k''a}).X''_{f''(1,k'',a)} \Bigg) \\ + \sum_{W \in \mathbf{V}} \left(\delta'_W \vee \delta''_W\right) : & \left(\gamma'_{iW} \vee \gamma''_{iW}\right) \to W \end{aligned} \tag{3.49}$$

provided

$$X'_1 = \sum_{a \in \mathcal{A}} \delta'_{1a} : \sum_{k' \in K'_{1a}} \gamma'_{1k'a} \to a(\mathbf{x}_{1k'a}).X'_{f'(1,k',a)} + \sum_{W \in \mathbf{V}} \delta'_{1W} : \gamma'_{1W} \to W \tag{3.50}$$

in E' and similarly X''_1 in E''. Call r_1 the right-hand side in equation (3.49). Similarly, call r'_1 and r''_1 the respective right-hand sides in equations for X'_1 in E' and X''_1 in E'' (see equation (3.50)). Using Lemma 3.2 and 3.3 , the reader should be able to show that $r_1 = r'_1 + r''_1$ from which this case is proved.

Case $t \equiv \mathbf{fix}Xt'$: By induction, t' satisfies a set of standard equations E' with free variables in $\mathbf{V}$, $X \in \mathbf{V}$. For every equation $X_i = r'_i$ in E' (definitions for E' are like in (3.48)) we define a new equation $X_i = r_i$ in E where each r_i is defined from r'_i by appropriately replacing variable X. For the distinguished variable X_1 we define:

$$X_1 = \sum_{a \in \mathcal{A}} \delta_{1a} : \sum_{k \in K_{1a}} \gamma_{1ka} \to a(\mathbf{x}_{1ka}).X_{f(1,k,a)} + \sum_{W \in \mathbf{V} - \{X\}} \delta_{1W} : \gamma_{1W} \to W$$

Call r_1 the right-hand side of the equation. Notice that r_1 is like r'_1 only that it omits the summand '$\mathbf{ff} : \mathbf{ff} \to X$' (since X does not occur unguarded in t', X must be guarded by predicate $\mathbf{ff}$). For $1 < i \leq |E'|$ we calculate the new equation as follows (calculations make use of Lemma 3.2 and 3.3).

$$X_i = r'_i[r_1/X]$$

$$
\begin{aligned}
&= \sum_{a\in\mathcal{A}} \delta_{ia} : \sum_{k\in K_{ia}} \gamma_{ika} \to a(\mathbf{x}_{ika}).X_{f(i,k,a)} \;+ \sum_{W\in\mathbf{V}-\{X\}} \delta_{iW} : \gamma_{iW} \to W \\
&\quad + \delta_{iX} : \gamma_{iX} \to \left(\sum_{a\in\mathcal{A}} \delta_{1a} : \sum_{k\in K_{1a}} \gamma_{1ka} \to a(\mathbf{x}_{1ka}).X_{f(1,k,a)} \;+ \sum_{W\in\mathbf{V}-\{X\}} \delta_{1W} : \gamma_{1W} \to W \right) \\
&= \sum_{a\in\mathcal{A}} \delta_{ia} : \sum_{k\in K_{ia}} \gamma_{ika} \to a(\mathbf{x}_{ika}).X_{f(i,k,a)} \;+ \sum_{W\in\mathbf{V}-\{X\}} \delta_{iW} : \gamma_{iW} \to W \\
&\quad + \sum_{a\in\mathcal{A}} ((\delta_{iX} \vee \delta_{1a}) \wedge \gamma_{iX} \wedge \textstyle\bigvee_{k\in K_{1a}} \gamma_{1ka}) : \displaystyle\sum_{k\in K_{1a}} (\gamma_{iX} \wedge \gamma_{1ka}) \to a(\mathbf{x}_{1ka}).X_{f(1,k,a)} \\
&\quad + \sum_{W\in\mathbf{V}-\{X\}} ((\delta_{iX} \vee \delta_{1W}) \wedge \gamma_{iX} \wedge \gamma_{1W}) : (\gamma_{iX} \wedge \gamma_{1W}) \to W \\
&= \sum_{a\in\mathcal{A}} (\delta_{ia} \vee ((\delta_{iX} \vee \delta_{1a}) \wedge \gamma_{iX} \wedge \textstyle\bigvee_{k\in K_{1a}} \gamma_{1ka})) : \\
&\qquad \sum_{k\in K_{1a}} (\gamma_{iX} \wedge \gamma_{1ka}) \to a(\mathbf{x}_{1ka}).X_{f(1,k,a)} + \sum_{k\in K_{ia}} \gamma_{ika} \to a(\mathbf{x}_{ika}).X_{f(i,k,a)} \\
&\quad + \sum_{W\in\mathbf{V}-\{X\}} (\delta_{iW} \vee ((\delta_{iX} \vee \delta_{1W}) \wedge \gamma_{iX} \wedge \gamma_{1W})) : (\gamma_{iW} \vee (\gamma_{iX} \wedge \gamma_{1W})) \to W
\end{aligned}
$$

Let t'_i, $i \in I$, be the set of terms that witnesses that t' $(= t'_1)$ satisfies E'. Noticing the $r_1 + \mathbf{ff} : \mathbf{ff} \to X \equiv r'_1$ the reader should not find difficulties on proving that the set of terms $t_i \equiv t'_i[t/X]$, $i \in I$, witnesses that t $(= t_1)$ satisfies E. (The proof needs REC). □

3.6.2 Completeness of the Proof System for Guarded Terms

Lemma 3.9 *For closed terms t and u, if $t \sim^{\phi} u$ then there exists ϕ' such that $\phi \Rightarrow \phi'$ and a standard closed equation set E which is provably ϕ'-satisfied by both t and u.*

Proof: Let the set of clock variables of t, u be $\mathbf{x}, \mathbf{y}$, respectively, with $\mathbf{x} \cap \mathbf{y} = \varnothing$. According to Lemma 3.8 let E_1 and E_2 be the standard closed equation sets for which t and u provably $\mathbf{tt}$-satisfy, respectively:

$$
\begin{aligned}
&E_1 : \{X_i = \textstyle\sum_{a\in\mathcal{A}} \delta_{ia} : \sum_{k\in K_{ia}} \gamma_{ika} \to a(\mathbf{x}_{ika}).X_{f(i,k,a)} \mid i \in I\} \\
&E_2 : \{Y_j = \textstyle\sum_{a\in\mathcal{A}} \delta'_{ja} : \sum_{l\in L_{ja}} \gamma'_{jla} \to a(\mathbf{x}_{ila}).X_{g(j,l,a)} \mid j \in J\}
\end{aligned}
$$

So there are $\tilde{t} = \{t_i \mid i \in I\}$ and $\tilde{u} = \{u_j \mid j \in J\}$ such that $t_1 = t, u_1 = u$, and

$$t_i = \textstyle\sum_{a\in\mathcal{A}} \delta_{ia} : \sum_{k\in K_{ia}} \gamma_{ika} \to a(\mathbf{x}_{ika}).t_{f(i,k,a)} \tag{3.51}$$

$$u_j = \textstyle\sum_{a\in\mathcal{A}} \delta'_{ja} : \sum_{l\in L_{ja}} \gamma'_{jla} \to a(\mathbf{x}_{ila}).u_{g(j,l,a)} \tag{3.52}$$

For each pair of i, j let $\Phi_{ij} = \{\omega \in \mathcal{RC}(\mathbf{xy}) \mid t_i \sim^{\omega\Uparrow} u_j\}$. Set $\varphi_{ij} = \bigvee \Phi_{ij}$. By the definition of Φ_{ij}, φ_{ij} is the weakest condition over which t_i and u_j are symbolically bisimilar, that is $\psi \Rightarrow \varphi_{ij}$ for any ψ such that $t_i \sim^{\psi} u_j$. In particular, $\phi \Rightarrow \varphi_{11}$. Also, for any $\omega \in \Phi_{ij}$ and $a \in \mathcal{A}$, $\omega \models \delta_{ia} \Leftrightarrow \delta'_{ja}$.

For each $a \in \mathcal{A}$ and $\omega \in \Phi_{ij}$, let

$$I_{ij}^{a\omega} = \{(k,l) \mid \omega \models \gamma_{ika} \wedge \gamma'_{jla} \text{ and } t_{f(i,k,a)} \sim^{\omega \downarrow_{\mathbf{x}_{ik}\mathbf{y}_{jl}} \Uparrow} u_{g(j,l,a)}\}$$

and define the set E containing equations

$$Z_{ij} = \textstyle\sum_{a \in \mathcal{A}} \delta_{ia} : \sum_{\omega \in \Phi_{ij}} \omega \to \sum_{(k,l) \in I_{ij}^{a\omega}} a(\mathbf{x}_{ika}\mathbf{y}_{jla}).Z_{f(i,k,a)g(j,l,a)}$$

We claim that E is provably φ_{11}-satisfied by t (resp. u) when each Z_{ij} is instantiated with t_i (resp. u_j) over φ_{ij}. We only prove the case of t; the case of u proceeds in a similar manner. For each i and j, we have to prove that

$$\varphi_{ij} \vdash t_i = \textstyle\sum_{a \in \mathcal{A}} \delta_{ia} : \sum_{\omega \in \Phi_{ij}} \omega \to \sum_{(k,l) \in I_{ij}^{a\omega}} a(\mathbf{x}_{ika}\mathbf{y}_{jla}).\varphi_{f(i,k,a)g(j,l,a)} \to t_{f(i,k,a)}$$

Because of (3.51) and soundness on the one hand, and CHOICE and Lemma 3.1.10 on the other hand, it suffices to prove the equivalence of each a-summand, that is, it suffices to prove that for all $a \in \mathcal{A}$,

$$\begin{aligned}\varphi_{ij} \vdash \textstyle\sum_{k \in K_{ia}} \gamma_{ika} \to a(\mathbf{x}_{ika}).t_{f(i,k,a)} = \\ \textstyle\sum_{\omega \in \Phi_{ij}} \omega \to \sum_{(k,l) \in I_{ij}^{a\omega}} a(\mathbf{x}_{ika}\mathbf{y}_{jla}).\varphi_{f(i,k,a)g(j,l,a)} \to t_{f(i,k,a)}\end{aligned}$$

Since the elements of Φ_{ij} are mutually disjoint, by Lemmas 3.1.1 and 3.1.18, it is sufficient to show that, for each $\omega \in \Phi_{ij}$,

$$\omega \vdash \textstyle\sum_{k \in K_{ia}} \gamma_{ika} \to a(\mathbf{x}_{ika}).t_{f(i,k,a)} = \sum_{(k,l) \in I_{ij}^{a\omega}} a(\mathbf{x}_{ika}\mathbf{y}_{jla}).\varphi_{f(i,k,a)g(j,l,a)} \to t_{f(i,k,a)}$$

By definition of $I_{ij}^{a\omega}$, we have that $t_{f(i,k,a)} \sim^{\omega \downarrow_{\mathbf{x}_{ika}\mathbf{y}_{jla}} \Uparrow} u_{g(j,l,a)}$. Hence, from the definition of $\Phi_{f(i,k,a)g(j,l,a)}$,

$$\omega \downarrow_{\mathbf{x}_{ika}\mathbf{y}_{jla}} \Uparrow \quad \Rightarrow \quad \varphi_{f(i,k,a)g(j,l,a)} \tag{3.53}$$

Under the assumption ω holds, we calculate,

$$\begin{aligned}
& \sum_{(k,l) \in I_{ij}^{a\omega}} a(\mathbf{x}_{ika}\mathbf{y}_{jla}).\varphi_{f(i,k,a)g(j,l,a)} \to t_{f(i,k,a)} \\
= & \ \{\text{By Lemma 3.1.17 from left to right}\} \\
& \sum_{(k,l) \in I_{ij}^{a\omega}} a(\mathbf{x}_{ika}\mathbf{y}_{jla}).\omega \downarrow_{\mathbf{x}_{ik}\mathbf{y}_{jl}} \Uparrow \to \varphi_{f(i,k,a)g(j,l,a)} \to t_{f(i,k,a)} \\
= & \ \{\text{Lemma 3.1.5 and (3.53)}\} \\
& \sum_{(k,l) \in I_{ij}^{a\omega}} a(\mathbf{x}_{ika}\mathbf{y}_{jla}).\omega \downarrow_{\mathbf{x}_{ik}\mathbf{y}_{jl}} \Uparrow \to t_{f(i,k,a)} \\
= & \ \{\text{By Lemma 3.1.17 from right to left}\} \\
& \sum_{(k,l) \in I_{ij}^{a\omega}} a(\mathbf{x}_{ika}\mathbf{y}_{jla}).t_{f(i,k,a)}
\end{aligned}$$

$$= \{\text{By THINNING}\}$$
$$\sum_{(k,l)\in I_{ij}^{a\omega}} a(\mathbf{x}_{ika}).t_{f(i,k,a)}$$
$$= \{\text{By Lemma 3.1.1 and SUBSTITUTION, since } \omega \models \gamma_{ika}\}$$
$$\sum_{(k,l)\in I_{ij}^{a\omega}} \gamma_{ika} \rightarrow a(\mathbf{x}_{ika}).t_{f(i,k,a)}$$
$$= \{\text{By claim below}^{2} \text{ using Lemma 3.1.15 and S1–S4.}(\dagger)\}$$
$$\sum_{k\in K_{ia}} \gamma_{ika} \rightarrow a(\mathbf{x}_{ika}).t_{f(i,k,a)}$$

It only remains to prove equality (†) and the proof of the lemma will be complete. For this, notice that $\{k \mid (k,l) \in I_{ij}^{a\omega}\} \subseteq K_{ia}$. Therefore, to prove (†), it suffices to show the following claim.

Claim 3.1 *If $k \in K_{ia} - \{k \mid (k,l) \in I_{ij}^{a\omega}\}$ then $\omega \Rightarrow \neg\gamma_{ika}$.*

Proof of claim. By contradiction suppose $\omega \Rightarrow \neg\gamma_{ika}$ is not the case, which is equivalent to say $\omega \Rightarrow \gamma_{ika}$ since ω is a region. Suppose $k \in K_{ia}$. Then $t \xrightarrow{a,\gamma,\delta,\mathbf{x}_{ika}} t_{f(i,k,a)}$ with $\omega \Rightarrow \gamma$. Besides, since $\omega \in \Phi_{ij}$, $t_i \sim^{\omega\Uparrow} u_j$. Then, there is a $(\omega\Uparrow \wedge\gamma)$-partition Φ s.t. for all $\phi \in \Phi$, $u_i \xrightarrow{a,\gamma',\delta',\mathbf{y}'} u'$, $\phi \Rightarrow \gamma'$ and $t_{f(i,k,a)} \sim^{\phi\Downarrow_{\mathbf{x}_{ika}\mathbf{y}'}\Uparrow} u'$. In particular this occurs for some ϕ s.t. $\omega \Rightarrow \phi$. Then, by soundness of equality, there must exist a summand $\gamma'_{jla} \rightarrow a(\mathbf{y}_{jla}).u_{g(j,l,a)}$ in (3.52), with $\mathbf{y}_{jla} = \mathbf{y}'$, $u_{g(j,l,a)} \sim^{\omega\Downarrow_{\mathbf{y}_{jla}}\Uparrow} u'$, and $\omega \Rightarrow (\gamma' \wedge \gamma'_{jla})$. But then $\omega \Rightarrow (\gamma_{ika} \wedge \gamma'_{jla})$, and hence, $(k,l) \in I_{ij}^{a\omega}$, by def. of $I_{ij}^{a\omega}$. □

Lemma 3.10 *If both t and u provably ϕ-satisfy an equation set E then $\phi \vdash t = u$.*

The proof of Lemma 3.10 proceeds as in Proposition 5.4 [LY02]. The completeness of the proof system is a direct consequence of Lemma 3.9 and Lemma 3.10:

Theorem 3.2 *For closed terms t and u, if $t \sim^{\phi} u$ then $\phi \vdash t = u$.*

Proof: Since $t \sim^{\phi} u$ and t and u are closed terms, by Lemma 3.9, exists a set of equations E which is ϕ'-satisfied by t and u and $\phi' \Rightarrow \phi$. By Lemma 3.10, $\phi' \vdash t = u$. Since $\phi \Rightarrow \phi'$, $\phi \vdash t = u$ by RESTRICTION. □

3.6.3 Completeness of the Proof System for all $\mathbb{A}$

In the following we show completeness for all closed terms. The strategy of proof is similar to [Mil89b] and it stands on the following lemma.

Lemma 3.11 *Let t be a term in which X occurs free and unguarded only outside the scope of a recursion. Then, there are predicates δ and γ and also a term u in which X does not occur unguarded such that $t = u + \delta : \gamma \rightarrow X$.*

Proof: The proof proceed by structural induction. For t having the form 0, $a(\mathbf{x}).t'$, $\mathbf{fix}Zt'$ (Z been X or any other variable), or a variable different from X, the lemma holds trivially. For the other cases we proceed as follows.

Case $t \equiv X$: Using axioms S1 and Lemmas 3.1.1 and 3.1.2, it is easy to show that $X = 0 + \mathbf{ff} : \mathbf{tt} \rightarrow X$.

Case $t \equiv t_1 + t_2$: By induction $t_i = u_i + \delta_i : \gamma_i \rightarrow X$, $i = 1, 2$. Using Lemmas 3.1.12 and 3.2, and axioms S3 and S4, it is possible to show that $t = u_1 + u_2 + ((\delta_1 \wedge \gamma_1) \vee (\delta_2 \wedge \gamma_2)) : (\gamma_1 \vee \gamma_2) \rightarrow X$.

Case $t \equiv \gamma \rightarrow t'$: By induction $t' = u' + \delta' : \gamma' \rightarrow X$. Then $t = \gamma \rightarrow u' + \delta : (\gamma \wedge \gamma') \rightarrow X$ by Lemmas 3.1.6 and 3.1.5.

Case $t \equiv \delta : t'$: By induction $t' = u' + \delta' : \gamma' \rightarrow X$. Then $t = \delta : u' + (\delta \vee \delta') : \gamma' \rightarrow X$ by Lemmas 3.1.9 and 3.1.8. □

Theorem 3.3 *For every term t there exists a guarded term t' s.t. $t = t'$ is provable.*

Proof: By induction we actually prove that for any t there is a t' s.t.

1. X is guarded in t';
2. no free unguarded occurrence of any variable Y in t' lies within a recursion t'; and
3. $\mathbf{fix}Xt = \mathbf{fix}Xt'$

from which the theorem follows. Suppose that 1, 2, and 3 hold for every u with recursion depth less than that of t. (The case when t contains no recursion follows in a similar manner.) Take a recursion $\mathbf{fix}Yu$ in t which lies within no recursion. By induction, there is a term u' s.t. Y is guarded in u', no free unguarded recursion of any variable lies within a recursion, and $\mathbf{fix}Yu = \mathbf{fix}Yu'$. Hence, no free unguarded occurrence of a variable occurs within a recursion in $u'[\mathbf{fix}Yu'/Y]$.

Let t_1 be the result of simultaneously replacing every top recursion $\mathbf{fix}Yu$ in t by $u'[\mathbf{fix}Yu'/Y]$. Clearly $t_1 = t$. Moreover, no free unguarded occurrence of a variable in t_1 lies within a recursion. By Lemma 3.11, there are predicates δ and γ, and t_2 in which X only occurs guarded, s.t. $t_1 = t_2 + \delta : \gamma \rightarrow X$. Then

$$\mathbf{fix}Xt = \mathbf{fix}Xt_1 = \mathbf{fix}X(t_2 + \delta : \gamma \rightarrow X) \overset{\mathsf{UR}}{=} \mathbf{fix}X(t_2 + \delta : \gamma \rightarrow t_2)$$

which proves the theorem. □

The following result is a consequence of Theorems 3.2 and 3.3.

Theorem 3.4 *For all closed $\mathbb{A}$ terms t and u, if $t \sim^{\nabla} u$ then $\phi \vdash t = u$.*

3.7 Conclusion

This chapter provides a sound and complete proof system for the coarsest congruence for (finite) timed automata with deadlines that is included in bisimulation.

The result on axiomatization can be extended to all $\mathbb{A}$ terms by noticing that ∇-bisimulation for open terms can be characterized either by extending the operational semantics allowing $X \xrightarrow{X} \mathbf{0}$ or by extending the symbolic semantics allowing $X \xrightarrow{\mathbf{tt},\mathbf{ff},X,\varnothing} \mathbf{0}$ for any variable X. The proof follows the lines of [Gla93].

By using standard ideas in [Mil89a, ABV94], it is possible to define axioms for static operations like hiding or parallel composition. Some operators have already been axiomatised in [BS00]. In particular, the following expansion law for parallel composition can be proved sound for the operational rules given in [DG05] ($\otimes$ is a 4-ary operation that returns a formula):

$$\begin{aligned} t \mathbin{||}^{\otimes}_{B} t' \;=\; & \sum_{i\in I, a_i \notin B} \delta_i : \gamma_i \to a_i(\mathbf{x}_i).(t_i \mathbin{||}^{\otimes}_{B} t') \;+\; \sum_{j\in J, b_j \notin B} \delta'_j : \gamma'_j \to b_j(\mathbf{y}_j).(t \mathbin{||}^{\otimes}_{B} t'_j) \\ & + \sum_{\substack{i\in I, j \in J,\\ a_i = b_j \in B}} ((\delta_i,\gamma_i)\otimes(\delta'_j,\gamma'_j)) : ((\gamma_i \wedge \gamma'_j) \to a_i(\mathbf{x}_i\mathbf{y}_j).(t_i \mathbin{||}^{\otimes}_{B} t'_j) \end{aligned}$$

where $t = \sum_{i\in I} \delta_i : \gamma_i \to a_i(\mathbf{x}_i).t_i$ and $t' = \sum_{j\in J} \delta'_j : \gamma'_j \to b_j(\mathbf{y}_j).t'_j$.

Acknowledgments

We thank Frits Vaandrager for his remarks on early drafts that helped to improve the quality of the present and the previous chapter. CONCUR'05/06 and FOSSACS'06 referees are also acknowledged for their useful remarks.

Chapter 4

Specifying Urgency in Timed I/O Automata

with Frits W. Vaandrager

Abstract Tools and techniques based on timed automata (such as Uppaal and the timed I/O automata framework) have proven to be extremely useful for the analysis of protocols and control software for real-time systems. However, a significant limitation of these approaches is that, due to the expressiveness of the modeling languages, timelocks — degenerate states in which time is unable to pass — can freely arise and cannot, in the general case, be detected. As a remedy to this problem Sifakis et al. advocate the use of *deadline* predicates for the specification of progress properties of Alur-Dill style timed automata. In this article, we extend these ideas to a more general setting, which may serve as a basis for deductive verification techniques. More specifically, we extend the TIOA framework of Lynch et al with *urgency* predicates. We identify a suitable language to describe the resulting *timed I/O automata with urgency* and show that for this language time reactivity holds by construction. We also establish that the class of timed I/O automata with urgency is closed under composition. The use of urgency predicates is compared with three alternative approaches to specifying progress properties that have been advocated in the literature: invariants, stopping conditions and deadline predicates. We argue that in practice the use of urgency predicates leads to shorter and more natural specifications than any of the other approaches. Some preliminary results on proving invariant properties of timed (I/O) automata with urgency are presented.

4.1 Introduction

In the literature on real-time systems there appears to be broad consensus on how to express quantitative timing constraints in state based modeling formalisms. Following the approach advocated by Alur and Dill [AD94], the idea is to designate certain state variables as clock variables. The values of these clock variables change as time advances. Also, clocks may be reset when discrete events occur. Timing constraints can be expressed, then, by conditions on clock values.

One issue on which there is no general consensus yet is how to specify progress properties, that is, properties which assert that a system must perform a certain action before a certain point in time. Merritt et al. in [MMT91] propose a model with upper and lower bounds associated with tasks (that is, sets of system actions). In the work of Alur and Dill [AD94], progress is enforced via a Büchi style acceptance criterion: by requiring that some (sets of) locations are visited infinitely often the possibility is ruled out that a system stays in certain locations forever. A popular approach, which is advocated in [HNSY94, AH94a] and implemented in the tool UPPAAL [LPY97], is to use (state) invariants. An invariant typically enforces a system action by limiting the amount by which time may advance in a given state. A related approach that is pursued in [KLSV03a] is to use stopping conditions. Here the idea is that when a system reaches a state in which a stopping condition holds, time may not progress any further and a system action has to occur immediately. Sifakis and his colleagues [BS00, SY96] advocate the use of deadlines for the specification of progress properties. Each transition of an Alur-Dill style timed automaton is decorated with an additional deadline predicate, which specifies when the transition becomes urgent. An advantage of the deadline approach (which can be viewed as a generalization of the approach of [MMT91]) is that under some reasonable assumptions, it ensures what is called time reactivity in [BS00] and timelock freedom in [Bow99], that is, whenever time progress stops there exists at least one enabled transition. Under certain conditions, time reactivity is even preserved by parallel composition of automata [Bow99, BS00, BGS00]. Similar to the original TIOA [KLSV03a], TIOA with urgency may still exibit *Zeno behaviour* – another type of timelock, where time is unable to pass beyond a certain point while actions continue to be performed [BG06],. The notion of deadlines has been incorporated in several modeling frameworks, see for instance [Bow99, GO01], and it has been implemented as part of the IF toolset [BGM02] and MoDeST [DHKK01].

The work of Sifakis et al [SY96, BS00] takes place in a setting of Alur-Dill style timed automata, a system model that has limited expressivity in order to enable automatic state space exploration and model checking. In this article, we study the specification of progress properties in the much more general model of timed I/O automata (TIOA) of Lynch et al [KLSV03a]. Even though fragments of the TIOA framework can be translated into timed automata [Rob04], analysis of general TIOA models requires the use of deductive verification techniques and theorem provers such as PVS [KLM04]. Inspired by the work of Sifakis et al, we introduce a similar notion of *urgency* predicates within the TIOA framework, both at the semantic level where we have infinite sets of states, transitions and trajectories, and at the syntactic level where system behavior is described finitely in terms of a logical language.

In the I/O automaton framework, transitions are typically specified using precondition/effect notation, that is, some type of guarded commands. This means that, for a given action name b with parameters $\vec{h}$ a precondition predicate $pre(\vec{v}, \vec{h})$ is given that defines from which states $\vec{v}$ action $b(\vec{h})$ is enabled, and an effect predicate $eff(\vec{v}, \vec{h}, \vec{v}')$ that defines to which states $\vec{v}'$ one may jump after doing action $b(\vec{h})$ in state $\vec{v}$. For the specification of timed systems we add a third predicate, the *urgency* predicate $urg(\vec{v}, \vec{h})$, to every transition definition. The meaning of the urgency predicate is that if, for some $\vec{h}$, the state predicate

$$pre(\vec{v}, \vec{h}) \wedge urg(\vec{v}, \vec{h}) \tag{4.1}$$

becomes true at a time point t in a trajectory, then t must be the limit time of that trajectory. Intuitively, the precondition specifies when a transition *may* occur, and the urgency predicate specifies when the transition becomes urgent, that is, either this or some other enabled discrete transition *must* occur immediately. A small but significant difference between our approach and the one of Sifakis et al [SY96, BS00] is that Sifakis et al require that a deadline predicate *implies* the precondition predicate, whereas we achieve a similar effect by *conjoining* the urgency predicate with the precondition.

The main contributions of this article are:

1. Extension of the work of [SY96, BS00, BGS00] on deadline predicates to a much more expressive setting, which may serve as a basis for deductive verification techniques. More specifically, we extend the TIOA framework of [KLSV03a] with urgency predicates at the semantic level, and define a suitable language to describe the resulting *timed I/O automata with urgency*. For this language, time reactivity holds by construction. We also establish that the class of TIOAs with urgency is closed under composition. In general, under the usual semantics, timed automata with urgency and timed automata with deadlines are not closed under composition, this problem has been studied in [DG05], where an alternative semantics is given that preserves compositionality.
2. A comparison of urgency predicates with three alternative ways to specify progress properties: invariants [HNSY94, AH94a], stopping

conditions [KLSV03a] and deadlines [SY96, BS00, BGS00]. Deadlines, stopping conditions and urgency predicates are shown to be (essentially) equally expressive. Invariants are slightly more expressive since they allow to bound the time at which an action occurs by a right open interval. Only use of urgency and deadline predicates gives time reactivity by construction. We argue that in practice the use of urgency predicates leads to shorter and more natural specifications than any of the other methods.

3. Some preliminary results on proving invariant properties of timed (I/O) automata with urgency. A similar approach for discrete time can also be found in [GB03].

The full version of the present paper appears as [GV04]. The proofs which have been omitted here, due to space limitation, are available in this technical report.

4.2 Timed (I/O) Automata with Urgency

In this section, we describe our extension of the timed I/O automata framework of Lynch et al [KLSV03a, KLSV03b] with urgency. In Subsections 4.2.1 and 4.2.2 we begin by recalling some definitions from [KLSV03a, KLSV03b]: we introduce a basic vocabulary for describing timed behaviors and recall the notion of a timed automaton. In Subsection 4.2.3, we add a notion of urgent transitions to timed automata, both at the semantic and at the syntactic level. The class of timed automata with urgency is not closed under composition, in general. In order to obtain compositionality, we add, in Subsection 4.2.4, an input/output distinction. Subsection 4.2.5, finally, defines a parallel composition operator and establishes that both the class of timed automata and the class of timed I/O automata with urgency are closed under composition.

4.2.1 Describing Timed System Behavior

In this section, we list the basic notions that are used in describing the behavior of a timed system, including both discrete and continuous changes. We simply sketch this material, leaving the reader to consult [KLSV03a, KLSV03b] for the details.

The time domain we use is the set R of real numbers (in [KLSV03a, KLSV03b] also other time domains are considered). States of automata will consist of valuations of *variables*. Each variable has both a *static type*, which defines the set of values it may assume, and a *dynamic type*, which gives the set of trajectories it may follow. We assume that dynamic types are closed under some simple operations: shifting the time domain, taking subintervals and pasting together intervals. We call a variable *discrete* if its dynamic type equals the pasting-closure of a set of constant-valued functions (i.e., the step-functions), and *analog* if its dynamic type equals the pasting-closure of a set of continuous functions (i.e., the piecewise-continuous functions).

A *valuation* for a set V of variables is a function that associates with each variable $v \in V$ a value in its static type. We write $val(V)$ for the set of all valuations for V. A *trajectory* for a set V of variables describes the evolution of the variables in V over time; formally, it is a function from a time interval that starts with 0 to valuations of V, that is, a trajectory defines a value for each variable at each time in the interval. We write $dom(\tau)$ for the domain of trajectory τ. A *point trajectory* is one with the trivial domain $\{0\}$. We write $\wp(\mathbf{x})$ for the point trajectory for valuation $\mathbf{x}$. The *limit time* of a trajectory τ, $\tau.ltime$, is the supremum of the times in its domain. $\tau.fval$ is defined to be the first valuation of τ, and if τ is right-closed, $\tau.lval$ is the last valuation. Suppose τ and τ' are trajectories for V, with τ closed. The *concatenation* of τ and τ', denoted by $\tau \frown \tau'$, is the trajectory obtained by taking the union of the first trajectory and the function obtained by shifting the domain of the second trajectory until the start time agrees with the limit time of the first trajectory; the last valuation of the first trajectory, which may not be the same as the first valuation of the second trajectory, is the one that appears in the concatenation. Trajectory τ is a prefix of trajectory τ', denoted $\tau \leq \tau'$, if τ can be obtained by restricting τ' to a subset of its domain. For every $t \in dom(\tau)$, we define $\tau \trianglerighteq t$ to be the trajectory obtained by taking the part of τ from t onwards, and then shifting the domain so that it starts with 0 again. Formally, $dom(\tau \trianglerighteq t) = \{u \in \mathsf{R} \mid u + t \in dom(\tau)\}$ and for all u in the domain, $\tau \trianglerighteq t(u) = \tau(u + t)$.

4.2.2 Timed Automata

A timed automaton in the sense of [KLSV03a, KLSV03b] is a state machine whose states are divided into *variables* and that has a set of discrete *actions*. The state of a timed automaton may change in two ways: by *discrete transitions*, which change the state atomically, and by *trajectories*, which describe the evolution of the state over intervals of time. Discrete transitions are labeled with actions, which are classified as either *external* or *internal*. The external actions are used to synchronize with the automaton's environment, while the internal actions are only visible to the automaton itself.

Formally, a *timed automaton* is a tuple $\mathcal{A} = (X, Q, \Theta, E, H, \mathcal{D}, \mathcal{T})$ with

- A set X of *internal variables*.
- A set $Q \subseteq val(X)$ of *states*.
- A nonempty set $\Theta \subseteq Q$ of *start states*.

- A set E of *external actions* and a set H of *internal actions*, disjoint from each other. We write $A \triangleq E \cup H$.
- A set $\mathcal{D} \subseteq Q \times A \times Q$ of *discrete transitions*. An edge $e = (\mathbf{x}, a, \mathbf{x}') \in \mathcal{D}$, also written as $\mathbf{x} \xrightarrow{a} \mathbf{x}'$, represent a transition from state $\mathbf{x}$ to $\mathbf{x}'$ labeled with action a. We say that a is *enabled* in $\mathbf{x}$ if $\mathbf{x} \xrightarrow{a} \mathbf{x}'$ for some $\mathbf{x}'$.
- A set $\mathcal{T}$ of trajectories for X such that $\tau(t) \in Q$ for each $\tau \in \mathcal{T}$ and $t \in dom(\tau)$. We require that the following axioms hold:

  T0 *(Existence of point trajectories)* If $\mathbf{x} \in Q$ then $\wp(\mathbf{x}) \in \mathcal{T}$.
  T1 *(Prefix closure)* For every $\tau \in \mathcal{T}$ and every $\tau' \leq \tau, \tau' \in \mathcal{T}$.
  T2 *(Suffix closure)* For every $\tau \in \mathcal{T}$ and every $t \in dom(\tau), \tau \trianglerighteq t \in \mathcal{T}$.
  T3 *(Concatenation closure)* Let $\tau_0\tau_1\tau_2\ldots$ be a sequence of trajectories in $\mathcal{T}$ such that, for each non final index, i, τ_i is closed and $\tau_i.lval = \tau_{i+1}.fval$. Then $\tau_0 \frown \tau_1 \frown \tau_2 \ldots \in \mathcal{T}$.

A trajectory τ is *maximal* in $\mathcal{T}$ if there exists no $\tau' \in \mathcal{T}$ with $\tau < \tau'$. The following lemma (which as far as we know is new) states that each trajectory can be extended into a maximal one. Intuitively this is an obvious property, but the proof requires some work due to the fact that we know so little about $\mathcal{T}$.

Lemma 4.1 *Let $\mathcal{A}$ be a timed automaton and let $\mathcal{T}$ be its set of trajectories. Then each trajectory in $\mathcal{T}$ is a prefix of a trajectory that is maximal in $\mathcal{T}$.*

Proof: Let $\tau \in \mathcal{T}$ be a trajectory. Suppose that τ can not be extended into a maximal trajectory. We derive a contradiction. Let $\tau_0 = \tau$, $t_0 = \tau_0.ltime$ and let u_0 be the supremum of the limit times of the trajectories in $\mathcal{T}$ that extend τ. Suppose $t_0 = u_0$. Since τ_0 is not maximal, this implies that it is right-open, and can be extended with a single state. But then the extended trajectory is maximal again, which is a contradiction. Hence $t_0 < u_0$. Let τ_1 be a trajectory that extends τ_0 such that if $u_0 = \infty$ then $t_1 = \tau_1.ltime = t_0 + 1$ else $t_1 = \tau_1.ltime > \frac{t_0+u_0}{2}$. Let u_1 be the supremum of the limit times of the trajectories in $\mathcal{T}$ that extend τ_1. Then $u_1 \leq u_0$. Continuing the construction, we find an infinite chain of trajectories $\tau_0 < \tau_1 < \tau_2 < \cdots$ and real numbers t_i, u_i such that $t_0 < t_1 < t_2 < \cdots u_3 \leq u_2 \leq u_1 \leq u_0$. In addition, we know that (1) there exists a k such that for all $j > k$, $u_j < \infty$ (otherwise it would be possible to extend τ_0 into an infinite and hence maximal trajectory), (2) $t' = \lim_{i\to\infty} t_i = \lim_{i\to\infty} u_i$. Let τ' be the limit of the trajectories τ_i. Then $t' = \tau'.ltime$. We know that $\tau' \in \mathcal{T}$ by axiom **T3** (concatenation closure). By assumption, since $\tau < \tau'$, τ' is not maximal in $\mathcal{T}$. Hence there exists a trajectory τ'' with $\tau' < \tau''$ and $t' < \tau''.ltime = t''$. But this contradicts the fact that, for sufficiently large k, the supremum of the limit times of the trajectories that extend τ_k is strictly less than t''. □

4.2.3 Adding Urgency

We now extend timed automata with extra state predicates, *urgency predicates*, one for each action.

A *timed automaton with urgency* is a pair $(\mathcal{A}, U)$ of a timed automaton $\mathcal{A} = (X, Q, \Theta, E, H, \mathcal{D}, \mathcal{T})$ and an *urgency predicate* $U : Q \times A \rightarrow Bool$. If $U(\mathbf{x}, a) = \mathbf{tt}$ then we say that action a is *potentially urgent* in a state $\mathbf{x}$. Action a is *urgent* in state $\mathbf{x}$ if it is potentially urgent and in addition enabled. We require that the following two axioms hold:

T4 *(Urgency)* For every $\tau \in \mathcal{T}$, $t \in dom(\tau)$ and $a \in A$: if a is urgent in $\tau(t)$ then $t = \tau.ltime$.

T5 *(Maximality)* For every $\tau \in \mathcal{T}$, if τ is maximal and finite then τ is right-closed and some $a \in A$ is urgent in $\tau.lval$.

Axiom **T4** states that as soon as an action becomes urgent, this action or some other action that is enabled has to occur immediately[1]. Axiom **T5** states that each maximal and finite trajectory enables an urgent action at the end.

Timed automata with urgency can be conveniently specified in a slight variation of the TIOA language [KLM04].

Example 4.1 *To illustrate this language, we consider the simple model of a train displayed in Figure 4.1. The automaton runs cyclically through states* $start$*,* $light$ *and* $gate$*. After spending between* $(2, 5]$ *time units in* $start$ *the automaton jumps to* $light$*, then within* $(5, 10]$ *time units after arrival in* $light$ *the automaton jumps to* $gate$*, and after exactly* 2 *time units in* $gate$ *the automaton returns to the initial state* $start$*.*

The definitions of the signature, state variables, initial states, and transition in our language are similar to their counterparts in the IOA language. We refer to the IOA user guide and reference manual [GLTV03] for additional information on this part of the language.[2] In this article, we consider only two types of state variables, which differ in their dynamic types: *discrete* variables (such as $control$), whose value remains unchanged along a trajectory, and *clocks* (such as x), which are real-valued variables whose value increases with rate 1 along a trajectory.[3] The set of states consists

[1]The reader may wonder why we do not impose a stronger axiom stating that if an action becomes urgent this action or some other *urgent* action has to occur immediately. Such an approach, which involves the use of priorities, has been studied in [GS05b]. It is well-known that priorities are incompatible with a trace based semantics. We feel that, for all practical purposes, axiom **T4** allows us to specify the desired urgency properties, while it is still fully compatible with the trace based semantics which has been the preferred semantic model for (timed) I/O automata since the first paper from 1987 [LT87].

[2]Since the emphasis in this article is on urgency we decided not to present all datatype definitions. At this point, our specifications are (deliberately) a bit sloppy.

[3]Our results easily generalize to more general dynamic types and continuous behavior defined by arbitrary differential equations and inclusions, such as studied e.g. in [KLSV03a, LSV03].

```
type $controlType$ = enumeration of $start$, $light$, $gate$
automaton $Train$
 states        $control : controlType$ initially $start$
               clock $x$ initially $0$
 signature     external $coming, approaching, passing$
 transitions   external $coming$
                 pre $x > 2 \wedge control = start$
                 urgent when $x \geq 5$
                 eff $control := light; x := 0$
               external $approaching$
                 pre $x > 5 \wedge control = light$
                 urgent when $x \geq 10$
                 eff $control := gate; x := 0$
               external $passing$
                 pre $x = 2 \wedge control = gate$
                 urgent when $true$
                 eff $control := start; x := 0$
```

Figure 4.1: A simple model of a train.

of all valuations of the state variables $\vec{v}$. At the syntactic level, we have a finite number of action names b and each action name comes with a list $\vec{h}$ of formal parameters. At the semantic level, the set of actions consists of pairs of an action name b and a valuation $\mathbf{h}$ of the parameters $\vec{h}$. The transition relation is defined via a finite number of transition definitions. Each transition definition consists of an action name b, a list $\vec{h}$ of formal parameters, a **pre**condition predicate $pre(\vec{v}, \vec{h})$ that defines from which states an action $b(\vec{h})$ is enabled, an **urgent when** predicate $urg(\vec{v}, \vec{h})$ that specifies when that action becomes urgent, and an **eff**ect predicate $eff(\vec{v}, \vec{h}, \vec{v}')$ specifying to which states $\vec{v}'$ one may jump after doing action $b(\vec{h})$ in state $\vec{v}$. If no parameters are mentioned, then the parameter list is assumed to be empty, if no precondition is mentioned then it is implicitly assumed to equal **tt**, and if no urgency predicate is mentioned this is assumed to equal **ff**. We further assume that the effect relation is total, in the sense that for each state $\mathbf{x}$ and parameter valuation $\mathbf{h}$ such that $pre(\mathbf{x}, \mathbf{h})$ holds, there exists at least one state $\mathbf{x}'$ such that $eff(\mathbf{x}, \mathbf{h}, \mathbf{x}')$ holds. If the effect predicate is defined using (deterministic) assignments, such as in Figure 4.1, this property trivially holds. The set of trajectories is defined implicitly. For $\mathbf{x}$ a state and t a non-negative real number, let $\mathbf{x} \oplus t$ be the state given by

$$\mathbf{x} \oplus t(v) \triangleq \begin{cases} \mathbf{x}(v) & \text{if } v \text{ is discrete} \\ \mathbf{x}(v) + t & \text{if } v \text{ is a clock.} \end{cases}$$

The state $\mathbf{x} \ominus t$ is defined similarly: replace $+$ by $-$ in the definition of $\oplus$. For $\mathbf{x}$ a state and I a time interval that starts with 0, a *pretrajectory* from $\mathbf{x}$

over I is a function $\tau : I \to Q$ such that for each $t \in I$, $\tau(t) = \mathbf{x} \oplus t$. The set of trajectories is defined to be the set of all pretrajectories τ satisfying that if some action a is urgent in some state $\tau(t)$, for t in the domain of τ, $t = \tau.ltime$.

Example 4.2 *Figure 4.2 gives another example of a specification in our language. It is a model of a reliable FIFO channel that delivers its messages within a certain time bound, represented by the automaton parameter b, which is a positive real number.*

The other automaton parameter M represents the type of messages communicated by the channel. The states of the automaton are valuations of the state variables queue and now. The discrete variable queue holds a finite sequence of pairs consisting of a message that has been sent and its delivery deadline. The clock variable now records the current real time. A send(m) action, which is always enabled and never becomes urgent, adds to the queue a new pair whose first component is m and whose second component is the deadline $now + b$. A receive(m) action can occur when m is the first message in the queue and it results in the removal of the first message from the queue. The receive(m) action becomes urgent when the delivery deadline u of the first message equals the current time now.

```
automaton Channel(b, M) where b∈R⁺
  states        queue ∈ (M × R)* initially empty
                clock now initially 0
  signature     external send(m), receive(m) where m ∈ M
  transitions   external send(m)
                  eff add (m, now + b) to the end of queue
                external receive(m)
                  pre ∃u : (m, u) is first element of queue
                  urgent when ∃u:(m, u)∈queue and now≥u
                  eff remove first element of queue
```

Figure 4.2: Time-bounded channel.

By construction, the set of trajectories denoted by a specification in our language satisfies axioms **T0-T4**. However, the example below shows that axiom **T5** does not need to hold in general.

Example 4.3 *The timed automaton specified in Figure 4.3 has a transition with precondition $x > 4 \wedge b = \textit{\textbf{ff}}$ and urgency predicate **tt**. Axiom **T5** does not hold, since time can only advance up to $x = 4$ but at that time the transition is not (yet) enabled.*

In order to avoid the counterexample of Figure 4.3, it is sufficient that certain predicates derived from the transition definitions are left-closed in the sense of [BGS00]. For each transition definition *tr*

```
automaton $\mathcal{A}$
  states        $b : Bool$ initially ff
                clock $x$ initially $0$
  signature     external $a$
  transitions   external $a$
                    pre $x > 4 \wedge b = \mathbf{ff}$
                    urgent when tt
                    eff $b := \mathbf{tt}$
```

Figure 4.3: A counterexample to axiom **T5**.

$$\begin{array}{ll} b(\vec{h}) & \textbf{pre } pre(\vec{v},\vec{h}) \\ & \textbf{urgent when } urg(\vec{v},\vec{h}) \\ & \textbf{eff } \mathit{eff}(\vec{v},\vec{h},\vec{v}') \end{array}$$

let predicate $Urg(tr)$ be given by

$$Urg(tr)(\vec{v},\vec{h}) \;\triangleq\; \exists \vec{h} : pre(\vec{v},\vec{h}) \wedge urg(\vec{v},\vec{h}) \tag{4.2}$$

Following [BGS00], we define a state predicate φ to be *left-closed* if, for all $\vec{v}$,

$$\neg\varphi(\vec{v}) \implies \exists \epsilon > 0\, \forall \epsilon' \leq \epsilon : \neg\varphi(\vec{v} \oplus \epsilon') \tag{4.3}$$

In practice, left-closedness can be easily obtained by only using non-strict lower bounds on clocks. For instance, $x \geq 4 \wedge b = \mathbf{ff}$ is left-closed but $x > 4 \wedge b = \mathbf{ff}$ is not. We can now formally state the following theorem.

Theorem 4.1 *If the predicate $\bigvee_{tr} Urg(tr)$ is left-closed then axiom* **T5** *holds.*

Proof: Suppose that τ is a maximal and finite trajectory. Assume that the domain of τ is right-open. Then, by **T4**, nowhere on τ an action becomes urgent. But this means that the extension of τ with a single state at the end gives a legal trajectory, thus contradicting the assumption that τ is maximal. Hence, without loss of generality, we may assume that the domain of τ is right-closed. Assume that no action a is urgent in $\tau.lval$. This means that the disjunction $\bigvee_{tr} Urg(tr)$ does not hold in **x**. But this means that there exists a small ϵ-extension of τ in which no action a is urgent. This extension is then a legal trajectory, which contradicts with the (assumed) maximality of τ. □

4.2.4 Adding an I/O Distinction

In this section, we further refine the model of timed automata by distinguishing between input and output actions as in [KLSV03a].

A *timed I/O automaton with urgency* is a quadruple $(\mathcal{A}, U, I, O)$ where

- $(\mathcal{A}, U)$ is a timed automaton with urgency, with $\mathcal{A} = (X, Q, \Theta, E, H, \mathcal{D}, \mathcal{T})$.
- I and O partition E into input and output actions, that is $E = I \cup O$ and $I \cap O = \varnothing$. Actions in $H \cup O$ are called locally controlled. We write $L \triangleq H \cup O$ and $A \triangleq E \cup H$.

We require that the following axiom holds:

E0 *(Inputs not urgent)* For every $\mathbf{x} \in Q$ and every $a \in I$, $U(\mathbf{x}, a) = \mathbf{ff}$.

E1 *(Input action enabling)* For every $\mathbf{x} \in Q$ and every $a \in I$, there exists $\mathbf{x}' \in Q$ such that $(\mathbf{x}, a, \mathbf{x}') \in \mathcal{D}$.

The input actions are assumed not to be under the automaton's control—they just arrive from the outside—while the automaton itself specifies what output and internal actions should be performed. In line with these intuitions, axiom **E0** states that input actions never become urgent. Axiom **E1** is the usual input enabling condition of ordinary I/O automata [LT87]; it says that a TIOA with urgency is able to accomodate an input action whenever it arrives. At the syntactic level, a sufficient condition for axioms **E0** and **E1** to hold is that, in each transition definition for an input action, the precondition is **tt** and the urgency predicate is **ff**.

A desirable property for models of real-time systems is *time reactivity*. This means that in each state, either time is allowed to advance forever, or time may advance for a while up to a point where the system is prepared to react with some locally controlled action. In [KLSV03a], an axiom **E2** is required for timed I/O automata which captures this property:

E2 *(Time-passage enabling)* For every $\mathbf{x} \in Q$, there exists $\tau \in \mathcal{T}$ such that $\tau.fval = \mathbf{x}$ and either

- $\tau.ltime = \infty$, or
- τ is right-closed and some locally controlled action $l \in L$ is enabled in $\tau.lval$.

For a TIOA with urgency, time reactivity is implied by the other axioms.

Theorem 4.2 *Each timed I/O automaton with urgency satisfies axiom* **E2**.

Proof: Assume that $\mathbf{x}$ is a state. Then, by axiom **T1**, $\wp(\mathbf{x})$ is a trajectory in $\mathcal{T}$. By Lemma 4.1 there exists a maximal trajectory τ that extends $\wp(\mathbf{x})$. By construction, $\tau.fval = \mathbf{x}$. By axiom **T5** either

- $\tau.ltime = \infty$ which completes the proof, or
- τ is right-closed and there is an action a that is urgent in $\mathbf{x}' = \tau.lval$. By axiom **E0** this cannot be an input action. Thus $\mathbf{x}'$ enables a locally controlled action. $\square$

4.2.5 Composition

We say that timed automata $\mathcal{A}_1$ and $\mathcal{A}_2$ are *compatible* if they have no state variables in common, and if neither automaton has an internal action that is an action of the other automaton. If $\mathcal{A}_1$ and $\mathcal{A}_2$ are compatible then their *composition* $\mathcal{A}_1||\mathcal{A}_2$ is defined formally to be the timed automaton $\mathcal{A} = (X, Q, \Theta, E, H, \mathcal{D}, \mathcal{T})$ where

- $X = X_1 \cup X_2$.
- $Q = \{\mathbf{x} \in val(X) | \mathbf{x} \lceil X_i \in Q_i, i \in \{1, 2\}\}$.
- $\Theta = \{\mathbf{x} \in Q | \mathbf{x} \lceil X_i \in \Theta_i, i \in \{1, 2\}\}$.
- $E = E_1 \cup E_2$ and $H = H_1 \cup H_2$.
- For each $\mathbf{x}, \mathbf{x}' \in Q$ and each $a \in A$, $\mathbf{x} \xrightarrow{a}_{\mathcal{A}} \mathbf{x}'$ iff for $i \in \{1, 2\}$, either (1) $a \in A_i$ and $\mathbf{x} \lceil X_i \xrightarrow{a}_i \mathbf{x}' \lceil X_i$, or (2) $a \notin A_i$ and $\mathbf{x} \lceil X_i = \mathbf{x}' \lceil X_i$.
- $\tau \in \mathcal{T} \Leftrightarrow \tau \downarrow X_i \in \mathcal{T}_i, i \in \{1, 2\}$.

We refer to [KLSV03b] for a proof that $\mathcal{A}_1||\mathcal{A}_2$ is a timed automaton, that is, the above structure satisfies axioms **T0**-**T3**.

Two timed automata with urgency, $(\mathcal{A}_1, U_1)$ and $(\mathcal{A}_2, U_2)$, are *compatible* if the underlying timed automata $\mathcal{A}_1$ and $\mathcal{A}_2$ are compatible. In this case, the *composition* is defined to be the structure $(\mathcal{A}, U)$, where $\mathcal{A} = \mathcal{A}_1||\mathcal{A}_2$ and U is given by

$$U((\mathbf{x}_1, \mathbf{x}_2), a) \quad = \quad U_1(\mathbf{x}_1, a) \vee U_2(\mathbf{x}_2, a),$$

where by convention $U_i(\mathbf{x}_i, a) = \mathbf{ff}$ if a is not in the signature of $\mathcal{A}_i$. So an action is urgent in a state of the composed system iff it is urgent in one of the component states. In general, the composition is not a timed automaton with urgency. The problem is due to axiom **T5**: if, for instance, we compose a system in which action a becomes urgent at time 1 with a system that has a in its signature but without any a-transition, then the composed system has a maximal trajectory of length 1 in which no transition is enabled. Several papers address the issue of how timelock freedom (or more generally, liveness) can be preserved by composition, see for instance [Bow99, BS00, BGS00]. In this article, we present one simple but useful result along these lines: the class of timed I/O automata with urgency *is* closed under composition.

We say that two timed I/O automata with urgency, $(\mathcal{A}_1, U_1, I_1, O_1)$ and $(\mathcal{A}_2, U_2, I_2, O_2)$, are *compatible* if the underlying timed automata $\mathcal{A}_1$ and $\mathcal{A}_2$ are compatible, and also they have no output actions in common. A consequence of these conditions is that each action is controlled by at most one component. In this case, the *composition* is defined to be the structure $(\mathcal{A}, U, I, O)$, where $(\mathcal{A}, U)$ is the composition of $(\mathcal{A}_1, U_1)$ and $(\mathcal{A}_2, U_2)$, $I = (I_1 \cup I_2) - (O_1 \cup O_2)$, and $O = O_1 \cup O_2$. That is, an external action of the composition is classified as an output if it is an output of one of the component automata, otherwise it is classified as an input.

Theorem 4.3 *The composition of two compatible timed I/O automata with urgency is again a timed I/O automaton with urgency.*

Proof: Straightforward from the definitions. Axiom **T5** holds because if τ is a maximal and finite trajectory of the composition there exists at least one component such that the projection of τ on that component is maximal. Using **T5** for the component gives that the projection is right-closed and that some action a of the component is urgent in the final state. By axiom **E0** we know that a is a locally controlled action. We infer that τ is right-closed and (by axiom **E1** for the other component) that a is urgent in the composed system in the final state of τ. $\square$

4.3 Expressivity

In this section, we compare the expressivity of urgency predicates with that of the deadline predicates of [SY96, BS00], the stopping conditions of [KLSV03a], and the invariants as used e.g. in [HNSY94, AH94a, LPY97].

4.3.1 Deadline Predicates

Instead of using urgency predicates, we could follow the approach of Sifakis et al [SY96, BS00] even more closely by using deadline predicates. This would mean that, for a given action name b with parameters $\vec{h}$, besides the precondition $pre(\vec{v}, \vec{h})$ and the effect $eff(\vec{v}, \vec{v}', \vec{h})$, also a *deadline* predicate $dl(\vec{v}, \vec{h})$ is specified such that $dl(\vec{v}, \vec{h}) \implies pre(\vec{v}, \vec{h})$ holds. The semantics of a deadline predicate is that if, for some $\vec{h}$, the state predicate

$$dl(\vec{v}, \vec{h}) \tag{4.4}$$

becomes true at a time point t in a trajectory, then t must be the limit time of that trajectory.

Clearly, any definition of a timed automaton with urgency predicates can be transformed into an equivalent (in the sense that the defined automata are semantically equal) definition with deadline predicates by replacing each urgency predicate $urg(\vec{v}, \vec{h})$ by a deadline predicate $pre(\vec{v}, \vec{h}) \wedge urg(\vec{v}, \vec{h})$. Conversely, any definition with deadline predicates can be transformed into an equivalent definition with urgency predicates by replacing each deadline predicate $dl(\vec{v}, \vec{h})$ by an identical urgency predicate $dl(\vec{v}, \vec{h})$. Studying the examples in Figure 4.1 and Figure 4.2, and the examples in [KLSV03b] indicates that the use of urgency predicates leads to slightly shorter specifications than the use of deadlines.

4.3.2 Stopping Conditions

Another alternative for urgency predicates are the stopping conditions as used in [KLSV03a]. A stopping condition is a state predicate $sc(\vec{v})$ such that if $sc(\vec{v})$ becomes true at a time point t in a trajectory, then t must be the limit time of that trajectory.

We again checked the examples from Figure 4.1, Figure 4.2 and [KLSV03b], and in each case urgency predicates lead to shorter and (in our view) more natural specifications than stopping conditions. Figure 4.4, for instance, shows how the transitions and trajectories of the example of Figure 4.1 can be rewritten using a stopping condition. The disadvantages should be clear: upper bounds are no longer specified next to the corresponding lower bounds, and parts of the preconditions have to be repeated in the stopping condition.

```
transitions    external coming
                 pre $x > 2 \wedge control = start$
                 eff $control := light; x := 0$
               external approaching
                 pre $x > 5 \wedge control = light$
                 eff $control := gate; x := 0$
               external passing
                 pre $x = 2 \wedge control = gate$
                 eff $control := start; x := 0$
trajectories   stops when
                     $(control = start \wedge x \geq 5) \vee$
                     $(control = light \wedge x \geq 10) \vee$
                     $(control = gate \wedge x = 2)$
```

Figure 4.4: The train model defined using a stopping condition.

Any definition of a timed automaton with urgency predicates can be transformed into an equivalent definition with stopping conditions by replacing the urgency predicates by a single stopping condition that is the disjunction of the formula $pre(\vec{v}, \vec{h}) \wedge urg(\vec{v}, \vec{h})$, for all transition definitions.

Stopping conditions are more expressive than urgency predicates since they allow one to define timed automata that are not time reactive and in which “the universe” may come to a halt. Figure 4.5 gives an example. Of

```
automaton $Doomsday$
  states         clock $x$ initially $0$
  trajectories   stops when    $x = 1$
```

Figure 4.5: A time deadlock.

course this is a form of additional expressivity that we would rather not have! For a timed automaton definition with a stopping condition $sc(\vec{v})$ it seems reasonable to require that the following variation of axiom **T5** holds:

T5′ *(Maximality)* For every $\tau \in \mathcal{T}$, if τ is maximal and finite then τ is right-closed, $sc(\tau.lval)$ and some (locally controlled) $a \in A$ is enabled in $\tau.lval$.

If this property holds, the specification can be transformed into an equivalent specification with urgency predicates: in case there is no I/O distinction we just add an urgency predicate $sc(\vec{v})$ to each transition definition, if there is an I/O distinction we add urgency predicate $sc(\vec{v})$ to each locally controlled transition and urgency predicate **ff** to each input transition.

4.3.3 Invariants

A popular way to specify progress properties, which has been advocated in [HNSY94, AH94a] and implemented in UPPAAL [LPY97], is the use of invariants. An *invariant* is a state predicate $inv(\vec{v})$ that is required to hold for all states along all trajectories. Figure 4.6 shows how the transitions and trajectories of the example of Figure 4.1 look with invariants. Again, like

```
transitions   external coming
                 pre $x > 2 \wedge control = start$
                 eff $control := light; x := 0$
              external approaching
                 pre $x > 5 \wedge control = light$
                 eff $control := gate; x := 0$
              external passing
                 pre $x = 2 \wedge control = gate$
                 eff $control := start; x := 0$
trajectories  invariant
                    $(control = start \wedge x \leq 5) \vee$
                    $(control = light \wedge x \leq 10) \vee$
                    $(control = gate \wedge x \leq 2)$
```

Figure 4.6: The train model defined with an invariant.

stopping conditions, invariants allow one to define timed automata that are not time reactive, a clear disadvantage of these specification styles. The example of Figure 4.5, for instance, can easily be encoded using invariants (replace the stopping condition by an invariant $x \leq 1$).

Invariants also allow one to specify strict upper bounds on the timing of events, as illustrated in Figure 4.7. The same timed automaton can not be specified using urgency predicates, for the simple reason that it has a maximal trajectory that is right-open, which is in violation of axiom **T5**. If

```
automaton BeforeOne
  states        discrete $b$ : $Bool$ initially ff
                clock $x$ initially $0$
  signature     external $a$
  transitions   external $a$
                  pre $b = \mathbf{ff}$
                  eff $b := \mathbf{tt}$
  trajectories  invariant $x < 1 \vee b = \mathbf{tt}$
```

Figure 4.7: Specification of a strict upper bound on timing with an invariant.

we are willing to consider timed automata up to some suitable equivalence (for instance, the trace equivalence defined in [KLSV03a]) then it is possible to specify strict upper bounds with urgency predicates, but this requires the use of auxiliary variables and unbounded nondeterminism. Figure 4.8 illustrates the specification of a strict upper bound with an urgency predicate. The idea is to choose nondeterministically a value in the interval $[0, 1)$ and then make *a* urgent when time has reached this value. Apart from the

```
automaton BeforeOne'
  states        $b$ : $Bool$ initially ff
                $t$ : R initially $0 \leq t < 1$
                clock $x$ initially $0$
  signature     external $a$
  transitions   external $a$
                  pre $b = \mathbf{ff}$
                  urgent when $x = t$
                  eff $b := \mathbf{tt}$
```

Figure 4.8: Specification of a strict upper bound on timing with urgency.

fact that the second specification is less intuitive, the use of unbounded nondeterminism will constitute a serious obstacle to automatic verification methods. In all practical applications of timed automata that we are aware of, the use of only non strict upper bounds on timing is not a restriction. For applications where use of strict upper bounds is essential, use of invariants is probably more appropriate than use of urgency predicates.

Timed automata with urgency predicates can (in many cases) be translated to equivalent timed automata with invariants. Robson [Rob04] describes how a fragment of TIOA with urgency predicates can be translated to the input language of UPPAAL.[4] Below we discuss a more general

[4]The UPPAAL syntax for invariant predicates is rather restricted. For each individual location the invariant is a conjunction of conditions of the form $x \leq e$ or $x < e$ where x is a clock and e is an expression that evaluates to an integer. This restriction forces Robson to

translation scheme. We say that a state predicate $\varphi(\vec{v})$ is *stable (under time progress)* if

$$\varphi(\vec{v}) \implies \forall d > 0 : \varphi(\vec{v} \oplus d) \tag{4.5}$$

Typically, a predicate will be stable if it only involves lower bounds on clocks and no upper bounds.

The *lower hull* of state predicate $\varphi(\vec{v})$ is the set of valuations given by

$$LH(\varphi) \triangleq \{\mathbf{x} \mid \varphi(\mathbf{x}) \wedge \exists \epsilon > 0\ \forall 0 < \epsilon' \leq \epsilon : \neg\varphi(\mathbf{x} \ominus \epsilon')\}$$

The upper hull of a state predicate can be defined similarly, just replace $\ominus$ by $\oplus$ in the above definition. If φ only involves (non-strict) lower bounds on clocks then the lower hull can easily be expressed again as a predicate by replacing the $\geq$ signs with $=$.

Now consider a definition of a timed automaton with urgency predicates such that all predicates $Urg(tr)$ are left-closed and stable. An equivalent timed automaton with invariants can be obtained by replacing the urgency predicates with the invariant

$$inv \quad = \quad \neg\left(\bigvee_{tr} Urg(tr)\right) \vee LH\left(\bigvee_{tr} Urg(tr)\right),$$

provided that the state predicate inv holds initially and after each discrete transition, i.e.,

$$pre(\vec{v}, \vec{h}) \wedge e\!f\!f(\vec{v}, \vec{h}, \vec{v}') \implies inv(\vec{v}').$$

The proof of the equivalence is straightforward and left to the reader.

Any timed automaton definition with right-closed invariants can be easily translated to a timed automaton with stopping conditions: the stopping condition is defined to be (a predicate denoting) the upper hull of the invariant. The translation scheme of Section 4.3.2 can then be used (provided axiom **T5′** holds) to translate the resulting timed automaton with stopping conditions to a timed automaton with urgency.

4.4 Proving Invariant Properties

In this section, we discuss how to establish invariant properties for specifications that involve urgency predicates. It is important to distinguish invariant properties from the invariant assertions that were discussed in the previous section as a construct to specify progress. An invariant property is a state predicate that *holds* for all reachable states of a given system.

split locations as part of her translation.

An invariant in the sense of previous section is an assertion that is actually used to *define* (the trajectories of) a system. Any invariant in the sense of the previous section is actually an invariant property of the system that it helps to define. The converse implication typically does not hold.

An *execution fragment* of a timed automaton $\mathcal{A}$ is a sequence $\alpha = \tau_0\, a_1\, \tau_1\, a_2\, \tau_2 \ldots$, where each a_i is an action of $\mathcal{A}$, each τ_i is a trajectory of $\mathcal{A}$, and for every i, $\tau_i.lval \xrightarrow{a_{i+1}} \tau_{i+1}.fval$. An execution fragment records what happens during a particular run of a system, including all the discrete state changes and all the changes that occur while time advances. An *execution* is an execution fragment whose first state is a start state of $\mathcal{A}$. A state is *reachable* in $\mathcal{A}$ if it is the last state of the last trajectory of a finite execution of $\mathcal{A}$. A state predicate φ is an *invariant* of $\mathcal{A}$ if it holds for all reachable states of $\mathcal{A}$.

In order to prove that an assertion φ is an invariant of $\mathcal{A}$, it suffices to prove that it holds initially and is preserved by all discrete transitions as well as by all $time(d)$ steps defined by

$$\begin{aligned} \mathbf{x} \xrightarrow{time(d)} \mathbf{x}' \;\; &\triangleq\;\; \exists \tau \in \mathcal{T} : \tau.fval = \mathbf{x} \;\wedge \\ &\;\;\; \tau.ltime = d \wedge \tau.lval = \mathbf{x}'. \end{aligned}$$

If we manage to give a simple and tractable characterization of the $time(d)$ predicate, then all the invariant proof techniques which are presented (for instance) in [MP95] become available in our setting.

Let tr be a transition definition for an action name b with parameter $\vec{h}$ with precondition $pre(\vec{v}, \vec{h})$, urgency predicate $urg(\vec{v}, \vec{h})$, and effect predicate $eff(\vec{v}, \vec{h}, \vec{v}')$. For $d \geq 0$, the *time progress* predicate $tp(\vec{v}, tr, d)$ expresses that transition tr permits time to advance with an amount d from state $\vec{v}$. The predicate is formally defined in terms of the $Urg(tr)$ predicate of (4.2):

$$\begin{aligned} tp(\vec{v}, tr, d) \;&\triangleq\; \forall 0 \leq e < d : \neg\, Urg(tr)(\vec{v} \oplus e, \vec{h}) \\ &\triangleq\; \forall 0 \leq e < d,\ \forall h : \\ &\;\;\; pre(\vec{v} \oplus e, \vec{h}) \implies \neg urg(\vec{v} \oplus e, \vec{h}) \end{aligned} \tag{4.6}$$

Using the time progress predicates, we characterize the time advance steps $time(d)$ as follows:

```
$time(d)$
    pre $\bigwedge_{tr} tp(\vec{v}, tr, d)$
    eff $\vec{v} := \vec{v} \oplus d$
```

In many cases it is possible to simplify the time progress predicates, by eliminating the universal quantifications from their definition. As an example, consider the timed automaton of Figure 4.1. The time progress predicate for the *coming* transition is

$$\forall 0 \leq e < d : \neg(x + e > 2 \wedge control = start \wedge x + e \geq 5)$$

$$\begin{aligned}
&\iff \forall 0 \le e < d : \neg(\mathit{control} = \mathit{start} \wedge x + e \ge 5)\\
&\iff \neg(\mathit{control} = \mathit{start} \wedge x + d > 5)\\
&\iff \mathit{control} = \mathit{start} \implies x + d \le 5
\end{aligned}$$

Similarly, the time progress predicates for the *approaching* and *passing* transitions can be written resp. as

$$\mathit{control} = \mathit{light} \implies x + d \le 10$$

$$\mathit{control} = \mathit{gate} \implies x + d \le 2$$

With these characterizations it is trivial to prove, for example, that

$$\mathit{control} = \mathit{start} \implies x \le 5$$

is inductive (and hence an invariant): it holds initially, and it is preserved by all discrete transitions and all $\mathit{time}(d)$ steps.

The above quantifier elimination can be generalized under some reasonable assumptions. If $\varphi(\vec{v})$ is a state predicate then we write $\mathit{post}(\varphi)(\vec{v})$ for the state predicate that holds for states that have a time predecessor satisfying φ:

$$\mathit{post}(\varphi)(\vec{v}) \quad \triangleq \quad \exists \vec{w}\, \exists e > 0 : (\vec{v} = \vec{w} \oplus e) \wedge \varphi(\vec{w}) \tag{4.7}$$

If φ only involves lower bounds on clocks, then $\mathit{post}(\varphi)$ can typically be obtained from φ by making these lower bounds strict, so quantifier elimination from $\mathit{post}(\varphi)$ is easy. Hence, if preconditions and urgency predicates only involve lower bounds on clocks (which appears to be a good specification style anyway), then their conjunction is stable. One may use the following lemma to eliminate the quantification over e from the time progress predicate (4.6).

Lemma 4.2 *Let φ be a state predicate that is stable under time progress. Then*

$$(\forall 0 \le e < d : \neg\varphi(\vec{v} \oplus e)) \Leftrightarrow \neg \mathit{post}(\varphi)(\vec{v} \oplus d) \tag{4.8}$$

Proof: Equivalence (4.8) can be rewritten into

$$(\exists 0 \le e < d : \varphi(\vec{v} \oplus e)) \Leftrightarrow \mathit{post}(\varphi)(\vec{v} \oplus d) \tag{4.9}$$

We prove both implications:

- $\Rightarrow$ Assume $\varphi(\vec{v} \oplus e)$, for certain $e \in [0, d)$. Then $\mathit{post}(\varphi)(\vec{v} \oplus d)$ holds since there is a state, namely $\vec{v} \oplus e$, that is a time predecessor of $\vec{v} \oplus d$ and in which φ holds.
- $\Leftarrow$ Assume $\mathit{post}(\varphi)(\vec{v} \oplus d)$. Then by (4.7) there exists an $e' > 0$ and a state $\vec{w}$ such that $\varphi(\vec{w})$ holds and $\vec{v} \oplus d = \vec{w} \oplus e'$. Depending on the relationship between d and e' we have two cases:

Case $e' \leq d$**:** Then $\vec{v} \oplus (d - e') = \vec{w}$. Choose $e = d - e'$. Then $d \in [0, d)$ and $\varphi(\vec{v} \oplus e)$.

Case $d < e'$**:** Then $\vec{v} = \vec{w} \oplus (e' - d)$. Since $\varphi(\vec{w})$ holds and φ is stable under time progress, also $\varphi(\vec{v})$ holds. So we may choose $e = 0$ to obtain $\varphi(\vec{v} \oplus e)$, as required. □

4.5 Concluding Remarks

In this article, we introduced a notion of urgency predicates and compared it with three other constructs for specifying progress properties that have been proposed in the literature: invariants, stopping conditons and deadlines. We showed that under some rather realistic assumptions (use of clock variables, no strict upper bounds on progress, absence of time deadlocks,...) the four notions are equally expressive. Nevertheless, a clear advantage of deadlines and urgency predicates in practice is that one gets absence of time deadlocks (time reactivity) for free. A potential advantage of invariants is that they allow one to bound the time at which a (locally controlled) action occurs by a right-open interval. However, we are not aware of practical applications in which this feature is really needed. We argued that if one uses a precondition/effect style specification language, urgency predicates lead to shorter and more natural specifications than any of the other constructs, in particular invariants. In the graphical syntax used by e.g. UPPAAL, the use of urgency/deadline predicates would not lead to shorter specifications than the use of invariants. Typically, in the first case one will decorate an edge of the graph (i.e., a transition) with a label $x \geq 4$, and in the second case a label $x \leq 4$ will be attached to a vertex of the graph (i.e., a location). But whereas the use of invariants may easily lead to time deadlocks, urgency/deadline predicates only stop time if there is a good reason for it, that is a specific transition that must be taken, and in this manner time deadlocks are avoided.

Folklore has it that urgency/deadline predicates are more difficult to implement in model checkers than invariants because they easily lead to non-convex zones. Non-convex zones indeed arise in the implementation of timed automata with deadlines in the IF toolset [BGM02]. In particular, time transitions may lead from one convex zone to several convex zones (not only one, as in standard timed automata with invariants). When such a situation arises in IF, the non-convex zone is automatically split into several, possibly overlapping, convex zones. The main reason why non-convex zones do not arise in standard timed automata with invariants (such as those implemented in UPPAAL) is that rather strong restrictions are imposed on the syntax of invariants. Only conjunctions of upper bounds on clocks—where the bounds are given by integer expressions—are allowed. If similar restrictions would be imposed in a syntax for urgency predicates, then no non-convex zones would arise in that setting either! More specifically, one would have to require that each urgency predicate is the disjunction of lower bounds on clocks, where the (non-strict) bounds are given by integer expressions. In addition, the urgency predicate of an input action *a*? should always be **ff**. We think it would be a clear improvement to the current version of UPPAAL (3.4.7) to add such a restricted notion of urgency predicates to the syntax, replacing the notion of an urgent channel. Adding general urgency predicates to UPPAAL would of course also be a possibility, but this would require splitting of zones as in the IF toolset.

In the setting that we studied, urgency predicates appear to be a very nice way to specify progress properties, with clear advantages over some other constructs that have been advocated in the literature. Some remaining questions for future research are:

1. Exploration of proof rules to reason with urgency predicates in simulations and liveness proofs.

2. Establish versions of the compositionality results of [Bow99, BS00, BGS00] in the setting of this paper.

3. Extension of our specification language and expressiveness results to a hybrid setting in which besides clocks also other continuously evolving variables are allowed.

Acknowledgements Thanks to Nancy Lynch and Dilsun Kaynar for detailled comments on an earlier version of this note, and to Marius Bozga for answering some of our questions about IF.

Chapter 5

A Formal Analysis of A Car Periphery Supervision System

with Tomas Krilavičius and Yaroslav S. Usenko

Abstract This paper presents a formal model of the real-time service allocation unit for the Car Periphery Supervision (CPS) system—a case study proposed by Robert Bosch in the context of the EU IST project AMETIST. The CPS system is a hybrid system, which is modeled in terms of timed automata. It is done by splitting the values of nonlinear continuous variables into finite sets of regions and over-approximating the constraints on continuous variables into clock constraints. Safety properties of the timed model have been verified using UPPAAL. This is a sufficient condition for validating the corresponding safety properties of the initial hybrid system. The difference in time scale between the CPS components have also been taken care of by over-approximating the timed model using the convex-hull over-approximation feature available in UPPAAL.

5.1 Introduction

A number of modeling and verification tools for real-time and hybrid systems have been developed. For instance, the tools based on the theory of *timed automata*, [AD94], such as KRONOS by [BDM$^+$98] and UPPAAL by [BDL$^+$01]; and the theory of *hybrid automata*, [Hen96], such as HYTECH by [HHWT97]. Recent developments in devising clever computational procedures have improved the ability of verification tools to handle industrial-size problems automatically. Yet many essential problems, specifically in the area of hybrid systems, remain unsolvable with these techniques.

Two major problems are (1) the lack of adequate abstraction concepts for modeling large systems, [AHLP00], (2) the exploding consumption of computing resources by the verification algorithms, [BCM$^+$92].

The present paper addresses the first issue and provides evidence that correct abstraction and appropriate over-approximation techniques in modeling of large systems leads to verifiable models from which one can infer properties of the original model.

This is illustrated by the real-time service allocation case study for Car Periphery Supervision (CPS) system (cf. [KR03]) proposed by Robert Bosch GmbH in the context of the EU IST project AMETIST. The CPS system is a hybrid system that interacts with a continuous environment via a discrete controller. Moreover, CPS safety properties are parametrized with free variables, which should be determined in order to prove the safety of the system. The CPS model presented in this paper is a timed automata based abstraction of the system constructed manually by dividing the environment into a finite set of regions.

Related Work

Verifying whether a hybrid system H satisfies a property P can turn out to be undecidable for most cases. Appropriate abstraction can extract a finite discrete system F from H by partitioning the state space of H into a finite number of regions. The survey by [AHLP00] aims to find a class of hybrid systems which can be abstracted into F and whose verification against a property P is decidable. The survey has shown that, proving that F satisfies P is equivalent or sufficient for proving that H satisfies P.

The approach used in the present paper is similar to the one of [HH95], where, nonlinear continuous variables of the original hybrid system are over-approximated to define a time-constrained automaton, and the approximated automaton will satisfy strictly fewer safety properties.

Another problem that leads to the state-space blowup in timed automata is the time scale difference between the different components of the system. This is often the case when an embedded system interacts with its environment (cf. [IKL$^+$00]). One way to solve such problems is to use the convex-hull over-approximation method of [HPR94] when verifying invariant properties.

Outline

The organization of the paper is as follows: Section 5.2 presents an informal description of the CPS system. Next, in Section 5.3, the CPS system is formally modeled in a way that the desired properties are fully preserved and the model of the system is abstracted to allow verification. Section 5.4 presents the properties and the verification results. Finally Section 5.5 concludes the paper and lists some directions for future work.

An abstract of this paper appeared as [GHKU03]. The complete UPPAAL model of the CPS system is available via the AMETIST project webpage.

5.2 Car Periphery Supervision System

Car Periphery Supervision (CPS) refers to the functionality and technology for obtaining information about the environment of a car. Applications like parking assistance, pre-crash detection and blind spot supervision depend on CPS for the basic operation and information sharing. Short Range Radar (SRR) sensors are mounted in-front of the car, and they scan the environment for nearby objects. The data collected by the sensors is sent to the computing part of CPS known as the Electronic Control Unit (ECU). The ECU processes the data and invoked applications based on the data. The structure of CPS and its environment is depicted in Fig. 5.1. A detailed description of the CPS system is given in [KR03], [Mor00] and [TFF+01].

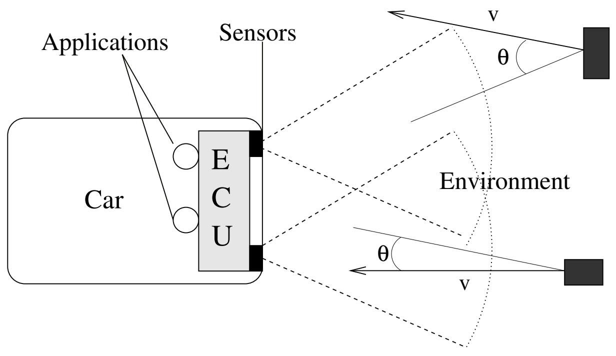

Figure 5.1: The CPS system and its environment

The CPS system and its environment can be viewed as a system consisting of three parts, namely:

1. **Environment:** the environment of the CPS system is a dynamic system with moving and stationary objects in-front of the car. Object velocity and distance are important characteristics that determine the behavior of the CPS system in a continuous manner. There are several restrictive assumptions that apply to the environment in which the CPS system is supposed to operate. This is done to make the system more tractable.

2. **Sensors:** the Sensors are the interfaces through which the CPS is informed about the behavior of the environment. Sensors operate in

discrete time, which can either be periodic or event-driven. The sensor component includes not only the equipment for sending and receiving radar signals but also a processor for basic data processing and control. Typically, the sensors that are situated in-front of the car return the distance to the nearest object (from their perspective) approaching the car.

3. **ECU:** the Electronic Control Unit is a board computer that performs a collection of tasks running on top of the OSEK operating system[1]. These tasks are used to control the operation of the sensors, and to deliver accurate and on-time information about the environment to the applications.

5.3 Formal Modeling

Each of the three parts of the CPS system are operating in different modes. The environment is a dynamic and *continuously* changing system. The sensors operate on a *discrete* time scale, while ECU tasks are *real-time* tasks. An appropriate model for this system would be a hybrid automaton. However, proving correctness of a system using hybrid automata is difficult, if at all possible.

Another approach is to abstract from the unnecessary details of the model and transform it into a timed automata model, while preserving the desired properties. In this section, the CPS system is modeled in timed automata, and relevant properties are verified using UPPAAL. This model consists of six timed automata that are put in parallel. The structure of the system is shown in Fig. 5.2. The boxes represent the timed automata,

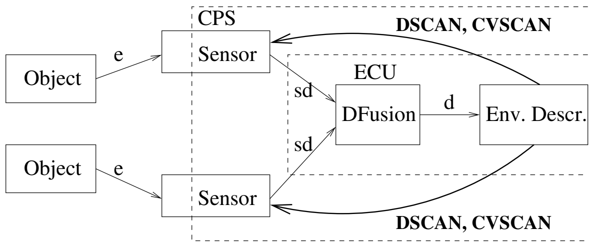

Figure 5.2: Decomposition of CPS in timed automata.

the thin arrows represent the communication via shared variables, and the thick arrows represent the multiparty action synchronization.

[1]`http://www.osek-vdx.org/`

5.3.1 CPS Environment

The environment is a collection of several objects moving with different velocities at different distances in-front of the car. An object i in the environment has the approaching velocity v_i and the angle θ_i relative to the direction of the car's movement (see Fig. 5.1).

Relative Velocity

Let $\mathbb{T} = \mathbb{R}^+$ denote the time domain and let d_i denote the distance to an object i from the middle of the front of the car. Then the relative velocity of an object is defined as the function $\dot{d}_i : \mathbb{T} \to \mathbb{R}$ and it is given by the difference between the velocity of the object and the velocity of the car (v_{car}):

$$\dot{d}_i = v_i \cos(\theta_i) - v_{car}$$

The CPS system only tracks the objects that are close enough to the car. The distances to the remote objects and their velocities do not affect the behavior of the CPS system.

Regions

As will be described in more detail in Section 5.3.2, the sensors in the CPS system scan for a nearest object, and return the distance to it. This can be given as:

$$d(t) = \mathsf{min}_{\forall i}(d_i(t))$$

The area in-front of the car is divided into twelve regions (see [KR03] for details). These regions are ordered in descending order and numbered from -1 to 9 (see Table 5.3.1).

Region name	Region number (e)
FAR	-1
PreCV	0
RG$_i$	1...8
PreCrash	9

Table 5.1: CPS regions

Assumptions

The environment that can be handled by the limited capacity of the sensors and other components of the CPS is rather small. The following three criteria restrict the behavior of the environment that the CPS system can interact with.

1. An object can approach the car with a relative velocity in the range from 13m/s to 56m/s.

2. Only one object is present in the environment.

3. The CPS system will be externally reinitialized when an object reaches the last region. In other words, the system terminates after the crash.

Environment behaviors outside the above assumptions have rare occurrence. Extra recovery treatments as in [KR03], which are not included in the current model, could be used to cope with such behaviors.

Timed Automaton of the Environment

Figure 5.3 shows a timed automaton model of the CPS environment, which is based on the allowed relative velocity defined above. The automaton has four locations which correspond to the regions names in Table 5.3.1. Initially objects are far from the car (in location `FAR` and `e:=-1`). If an object comes closer to the car it will reach the location `PreCV`.

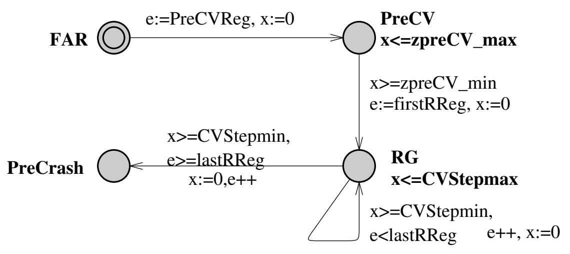

Figure 5.3: CPS environment template

The constants `zpreCV_max` and `zpreCV_min` are time bounds on the transition of the environment from `PreCV` to $\texttt{RG}_0$. These time bounds are calculated from velocity bounds and the length of the `PreCV` region. Other time bounds shown in Fig. 5.3 are calculated in a similar manner. This method of substituting the maximal and minimal velocity constraints into clock constraints is done in accordance with the method of [HH95].

An object has to cross all the regions up-to the `PreCrash` region. (`e` $\geq$ `lastRReg`) before it can go into `PreCrash`. As soon as the object is in `PreCrash`, an immediate action has to be taken by the ECU. The clock `x` is reset to measure the amount of time that the object spends in the `PreCrash` location before the ECU does something about it. Section 5.4 lists important invariants that have to be satisfied when an object reaches the `PreCrash` location.

5.3.2 Sensor

A sensor in the CPS system scans the environment for a nearest object in-front of the car, and returns the region value of the scanned object. A sensor has two modes of operation: `DSCAN` and `CVSCAN`, and one `IDLE` mode. These modes of operation are used as locations in the timed automaton of the sensor depicted in Fig. 5.4.

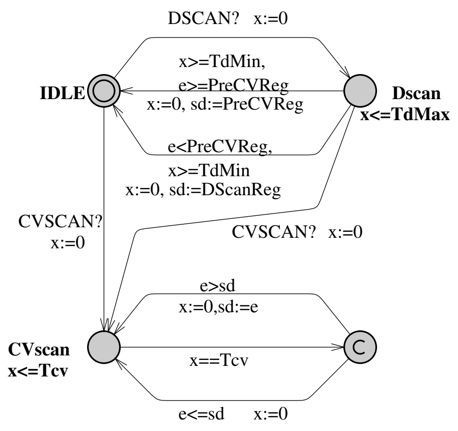

Figure 5.4: Sensor Template

The sensor is initially in location `IDLE` and waits for a command from the ECU. If it receives a `DSCAN` command, it conducts a long range scan and, as soon as it finds an object in `PreCV`, the sensor's distance reading `sd` is updated:

$$sd_i(t) = e_i(t),$$

where t is the time when the ith sensor scans the environment from its own perspective (e_i). For a `CVSCAN` command, the sensor produces a maximum of eight readings, one for each region $\mathtt{RG}_i$. It scans the environment every `Tcv` time units, and sd_i is updated in a similar manner.

The CPS system usually has several sensors placed in-front of the car. In this paper only two sensors are considered. It is possible that the two sensors return different readings. This may be a result of the fact that they are situated at different positions, or the object may have an irregular form, or simply there are several objects in-front of the car. Thus, in general, there do not need to be correlations between the readings of the two sensors (even though a visibility analysis shows that the difference is rather small).

5.3.3 Electronic Control Unit (ECU)

ECU stands for the collection of tasks running on a single processor. According to [KR03], the ECU executes more than two tasks. Most of these tasks are sequential and run in a predictable manner. Thus, they can easily be grouped into two tasks without affecting their behavior. These combined tasks are called DFusion (sensor fusion) and EnvDescription (environment description).

Sensor Fusion

DFusion is a part of the ECU which receives individual distance values from all sensors and sends a "combined" single value to the remaining tasks. Combining several readings (sd_is) into one reading (d) can be done by the triangulation function as suggested by [KR03]. The maximum function, instead of triangulation, is used here to compute the final value, since our model deals with one-dimensional value of the sd_is only. DFusion is a periodic task and in every jth step it computes $d(j)$ as follows:

$$d(j) = \mathsf{max}_{\forall i=\{1,2\}}(sd_i(j))$$

The variable $d(j)$ is a shared variable which is also readable by EnvDescription. Figure 5.5 shows the timed automaton of DFusion.

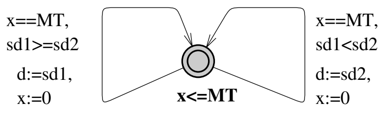

Figure 5.5: Sensor Fusion

Environment Description

EnvDescription is a part of ECU that receives environment data from DFusion and maintains accurate information about the environment. While doing so, EnvDescription controls the mode of operation of the sensors as well. The sensor-controlling part is defined in [KR03] as the "situation analysis" task. In the present paper it is combined with the EnvDescription task to avoid the state-space explosion.

The timed automaton of EnvDescription is shown in Fig. 5.6. Initially EnvDescription is in location `IDLE`, from which it periodically reads the value of `d` and broadcasts the `DSCAN` command to the sensors. If the value of `d` shows that an object is present in the `PreCV` region, then the automaton jumps to location (`PreCV0`), from which it sends the `CVSCAN` command.

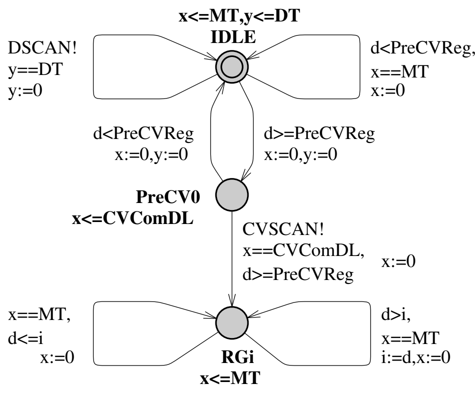

Figure 5.6: Environment Description (ED)

There is a time delay of `CVComDL` time units before this command happens. Once the sensors receive the `CVSCAN` command, the EnvDescription counts the number of $\mathtt{RG}_i$ regions by reading `d` computed by DFusion.

5.4 Verification

5.4.1 Requirements

The primary goal of the CPS system is to provide accurate information about the environment of the car to the applications such as airbag inflation, parking assistance, pre-crash detection and others. The accuracy of the information provided by ECU is measured by the time delay between a change in the environment and the knowledge of the ECU about this change. The properties are stated below as parametrized temporal logic formulas, and they were verified using UPPAAL on a workstation with a 512MHz CPU and 256Mb of RAM. The time required to verify these properties was drastically reduced using the convex-hull approximation feature of UPPAAL to less than a minute.

Property 1 (P1): EnvDescription has an accurate information about the position of the object in the collision course. This property is modeled in the following way: the EnvDescription's information `ED.i` about the region of the object should not deviate too much from the environment's information `e1` about the same object. The goal is to find the maximal difference `Q` between these two values (the minimal Q for which the formula is satisfied).

```
A[] (e1-ED.i <= Q and e2-ED.i <= Q)
```

Property 2 (P2): When an object reaches the pre-crash region (the environment automaton is in location `ENV1.PreCrash`), EnvDescription knows about this (`ED.i == lastRReg+1`) within a few time units (`ENV1.x > P`). Here `ENV1.x` represents the time after the environment automaton moved into location `ENV1.PreCrash`, and we are interested in finding the minimal value of `P`.

```
A[] ((ENV1.PreCrash and ENV1.x > P)
     imply (ED.i == lastRReg+1))
```

Property 3 (P3): The ECU avoids false alarm. EnvDescription never reports advancement of an object toward the car before the object (the environment) actually does so.

```
A[]  (ED.i <= e1 or ED.i <= e2)
```

Property 4 (P4): The system is deadlock free.

```
A[] (not deadlock)
```

5.4.2 Results

The CPS model satisfies properties `P1` for $Q \geq 3$, `P2` for $P \geq 5$, `P3` and `P4`. Note that Q can also be computed from P as $Q = \lceil P/CVStepmin \rceil$, since the difference in the number of regions can also be expressed as a difference in time.

In the above model DFusion and EnvDescription run in arbitrary order, no prior scheduling is assumed. It is possible, however, to schedule the execution in such a way that, DFusion computes `d` and EnvDescription updates its counter immediately. Under such synchronization EnvDescription will update its information faster. Figure 5.7 shows the model of EnvDescription using this alternative schedule. The model of the new DFusion is not shown here, but it is similar to the one in Fig. 5.5 except that the new model sends a synchronization signal (`ND!`) to the EnvDescription as soon as a new value of `d` is computed. After the synchronization with DFusion, EnvDescription makes its decision without a delay (see the urgent locations

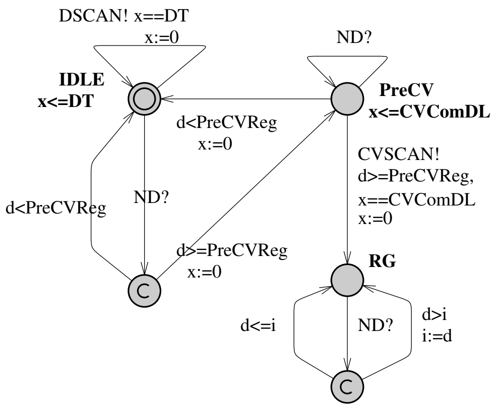

Figure 5.7: Modified Environment Description

in Fig. 5.7). For this setting the properties P1 for $Q \geq 2$ and P2 for $P \geq 3$ are satisfied.

In both cases P is equal to the time needed for an information to propagate from the environment to EnvDescription. That is the sum of the time spent by the sensor (Tcv), DFusion and EnvDescription.

$$P = Tcv + MT + MT$$

But in the scheduled model, both EnvDescription and DFusion need only one MT to update their information, and propagation time is reduced to $P = Tcv + MT$. These two cases show the worst and best cases for determining the value of P. In general, however, P is equal to $Tcv + MT$ plus the overhead associated with scheduling of the ECU tasks and the time spent by each task. When OSEK operating system, is used to schedule the ECU tasks, and if $OSEK_{SchTime}(T_i)$ is the time delay due to scheduling and running the tasks T_i, then the value of P is

$$P = Tcv + MT + OSEK_{SchTime}(T_i).$$

5.5 Conclusion and Future Work

The car periphery supervision is a hybrid system. Verifying properties for hybrid systems is undecidable in general. However, the continuous variables of the model, the environment in this case, can be discretized to a finite block of regions in order to make verification of the properties possible.

The different time scale between the environment and CPS components have resulted in a state-space blow up. The convex-hull over-approximation technique of UPPAAL was used to verify the safety properties of the system. Another approach could be the exact acceleration method of [HL02].

The assumptions made on the environment of CPS are too restrictive. Some of them can be omitted by introducing a recovery mode in case when more than one object appears in the `RG` regions. As described in [KR03], the recovery mode is a third mode of sensor operation, which scans for follow-up objects when a nearest object disappears from the scene. This scenario may happen when one object in the collision course changes its trajectory and disappears; and later on, another object, which was close to the first one, enters a collision course. Adding a recovery method to the model for such a scenario would be an interesting future step.

Acknowledgments

The authors would like to thank (in reverse-alphabetic order) Frits Vaandrager, Stefan Kowalewski Holger Hermanns, Marko Auerswald and the members of the Sensor Group at Robert Bosch GmbH for their valuable comments and reviews.

Chapter 6

Control Synthesis for a Smart Card Personalization System using Symbolic Model Checking

with Frits W. Vaandrager

Abstract Using the Cadence SMV symbolic model checker we synthesize, under certain error assumptions, a scheduler for the smart card personalization system, a case study that has been proposed by Cybernetix Recherche in the context of the EU IST project AMETIST. The scheduler that we synthesize, and of which we prove optimality, has been previously patented. Due to the large number of states (which is beyond 10^{13}), this synthesis problem appears to be out of the scope of existing tools for controller synthesis, which typically use some form of explicit state enumeration. Our result provides new evidence that model checkers can be useful to tackle industrial sized problems in the area of scheduling and control synthesis.

6.1 Introduction

Background

Model checking involves analyzing a given model of a system and verifying that this model satisfies some desired properties. System models are typically described as finite transition systems, while properties are described in terms of temporal logic. Once the definition of the system, **S**, and its property, ψ, are fixed, the model checking problem is easily described as $\mathbf{S} \models \psi$? (does **S** satisfy ψ?). Thanks to the symbolic representation of transition systems, state-of-the-art model checking tools are now capable of solving such problems for models with more than 10^{20} states [BCM$^+$92].

Control synthesis, on the contrary, does not assume the existence of a model of the full system. Instead, it considers the uncontrolled plant and tries to synthesize a controller by finding a possible instance of a model that satisfies a desired property. Control synthesis for Discrete Event Systems (DES) has been extensively studied over the past two to three decades, and a well-established theory has been developed by Ramadge and Wonham [RW89]. The Ramadge and Wonham framework (RW) is based on the formal (regular) language generated by a finite state machine. The RW plant model P (*generator*) is obtained by describing the plant processes in terms of a formal language which is generated by a finite automaton. A *means of control* is adjoined to this *generator* by identifying the events that can be enabled or disabled by the controlling agent. The specifications S_p are described in terms of the formal language generated by P. The controller is then constructed from a recognizer for the specified language given by S_p.

Control synthesis problems for Discrete Event Systems like the Cybernetix smart card personalization system [Alb02] are covered by the Ramadge and Wonham supervisory control theory. In the present paper, however, we (partially) solve the problem using a model checker, namely SMV [McM93].[1] This approach allows us to benefit from the (BDD-based) symbolic representation technique of SMV and to solve the problem which, because of its size (more than 10^{13} states), would be intractable otherwise. Our results demonstrate that model checkers can be useful to solve problems in the area of scheduling and control synthesis.

Outline

Using SMV we synthesize a scheduler for a smart card personalization system, which has previously been patented by Cybernetix Recherche. We also show that this scheduler, known as the "super single mode" [Alb02] is optimal in the absence of errors. Finally, we synthesize a set of schedulers for defective card treatment that stabilize the system back to the super single mode.

The paper is structured as follows: Section 6.2 provides a formal definition of the uncontrolled plant of the smart card personalization system, and defines the correctness and optimality criteria. Section 6.3 explains the super single mode, and how it was generated using SMV. Section 6.4 deals with systems with faulty cards. We list the errors that may occur during the operations of the machine, show how to deal with such errors, and give

[1]We use the version of SMV developed at Cadence Berkeley Laboratories, see `http://www-cad.eecs.berkeley.edu/~kenmcmil/smv/`.

an overview of the synthesized error treatment methods. We conclude the paper by pointing out some observations and directions for future work in Section 6.5. Extended version of this work appears in KUN technical report [GV03b]. An electronic copy of SMV code and also of trace simulator that we developed to visualize the schedules are available via the URL

`http://www.cs.kun.nl/ita/publications/papers/biniam/cyber`.

Related Work

The Ramadge and Wonham framework has been implemented by several research groups and industries. One of the tools developed by Wonham and his research team is CTCT (C based Toy Control Theory)[2], a tool that was basically built for research purposes only, and uses an exhaustive list to represent the model. Its capacity, as the name indicates, has never extended beyond toy examples. A new approach, Vector Discrete Event Systems, was studied in [LW93, YL94] to alleviate the shortcoming of CTCT by exploiting the structural properties of DES. Although this approach resulted in better performance, its structural analysis approach cannot be generalized [CL99].

Other notable developments on this area are: The UMDES-LIB library from University of Michigan [SSL$^+$95], Bertil Brandin's tool for DES control synthesis with heuristics [Bra96], a tool for Condition/Event Systems [SK91], other tool by Martine Fabian and Knut Åkesson [ÅF99].

All the above tools lack symbolic representation of state transitions, and suffer from state space explosion problems. A Binary Decision Diagram (BDD) like data structure called Integer Decision Diagram (IDD) has been used to represent sets of states symbolically. For example, Gunnarsson in [Gun97] and Zhang and Wonham in [ZW01] have used IDDs in their implementation. This approach is quite promising for dealing with large systems, but it is still in laboratory stage, and not available to the public.

Our main motivation for using SMV is thus to overcome this deficiency and benefit from symbolic representation of SMV. The smart card personalization system is quite a large system and cannot be handled with a tool that does not use symbolic representation. Our paper shows how the scheduler synthesis can be solved using a model checker and presents new evidence that model checkers can be useful in solving problems in the area of scheduling and synthesis. A similar technique was also employed in [Feh99, HLP01, NY00] to synthesis scheduler for industrial size problems.

We were the first to model the smart card personalization system and to synthesize a scheduler for it. However, the same case study has also been addressed by other members of the AMETIST consortium. T. Krilavicius

[2]See `http://odin.control.toronto.edu/people/profs/wonham`.

and Y. Usenko using UPPAAL and μCRL [KU03], T. Ruys in SPIN [Ruy03], A. Mader using UPPAAL [Mad04] and G. Weiss in Life Sequence Charts (LSC) [Wei03] have studied the same problem. But none of the mentioned approaches deals with error handling.

6.2 Smart Card Personalization System

The “smart card personalization system” is a case study that has been proposed by Cybernetix Recherche in the context of the EU IST project AMETIST [Alb02]. The case study concerns a machine for smart card personalization, which takes piles of blank smart cards as raw material, programs them with personalized data, prints them and tests them.

The machine has a throughput of approximately 6000 cards per hour. It is required that the output of cards occurs in a predefined order. Unfortunately, some cards may turn out to be defective and have to be discarded, but without changing the output order of personalized cards. Decisions on how to reorganize the flow of cards must be taken within fractions of a second, as no production time is to be lost.

The goal of the case study is to model the desired production requirements as well as the timing requirements of operations of the machine, and on this basis synthesize the coordination of the tracking of defective cards. More specifically, the goal is to synthesize optimal schedules for the personalization machine in which defective cards are dealt with, i.e., schedules in which

1. cards are produced in the right order (safety). The order of cards is important as no other sorting mechanism should exist in the system,

2. throughput is maximal (liveness).

6.2.1 The Uncontrolled Plant Model

Figure 6.1 shows a simplified smart card personalization machine. The machine consists of a conveyor belt and personalization stations mounted on top of it. The machine also has an input station and an output station, which are situated on the left and right side of the belt respectively. New cards enter the system through the input station and advance to the right one step at a time. At some point, a card is lifted up to one of the personalization stations, spends some time there (is personalized), and is then dropped back onto the belt. The card then moves towards the output station for testing and delivery. The actual machine is considerably more complicated than the machine in Figure 1, but our aim is to find a scheduler that effectively utilizes the personalization stations and optimizes throughput.

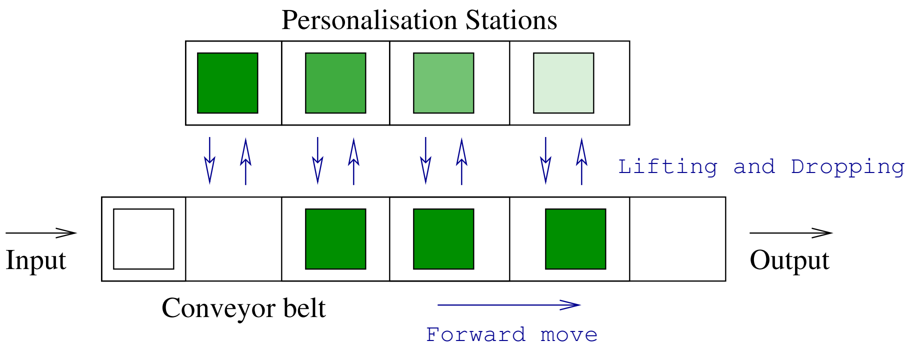

Figure 6.1: Simplified smart card personalization machine.

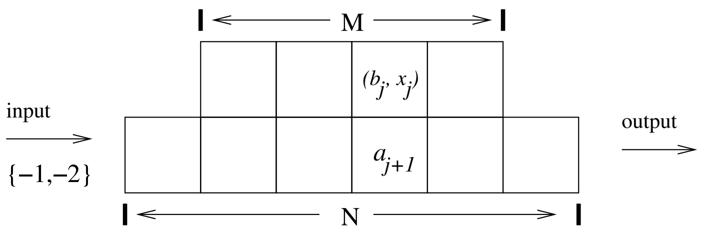

Figure 6.2: The model of the smart card personalization machine.

The simplified model of the machine appears to be adequate for this purpose.

The SMV model for the uncontrolled machine is a collection of processes running concurrently: `forward` (moving a belt one step to the right) and, for each personalization station j, $\texttt{lift\_drop}_j$ (lifting/dropping a card from/to the belt to/from station j). We employ a discrete model of time, in which one time unit is equivalent to one forward move of the belt. All personalization stations are identical and need `S` time units to personalize a card. We assume lifting and dropping takes no time.

We assume there are `M` stations (denoted by $\texttt{b}_j$), and `N = M+2` slots in the belt (denoted by $\texttt{a}_j$) as shown in Figure 6.2. To make model checking possible, the number of different personalizations is assumed to be bounded by some value `K`, which is a multiple of `M`. Each slot or station will have a value as shown in Table 6.1. An empty slot/station is coded twice (as `-3` and `-2`) in order to distinguish between the initial value (`-3`) and the slot/station being emptied along the way (`-2`). This allows us to control intermediate blank slots more efficiently, as will be explained below. We also use an integer variable $\texttt{x}_j$, ($\texttt{0} \leq j$ `<` `M`) as a clock to record how long a card has been held in station j.

Formally, the process `forward` is defined as follows.

Table 6.1: System parameters and encoding of values.

parameter	represents
M	number of stations
N	total number of slots
K	different number of personalizations
S	time needed for personalization

slot/station value	meaning
-3	empty (initial value)
-2	emptied
-1	new card
$j, 0 \leq j < K$	personalized with j
K	defective card

```
module forward(a,b,x){

  next(a[0]):={-1,-2};           /* a new card appears    */
  for(j=1;j<=N-1;j=j+1)          /*non-deterministicaly   */
         next(a[j]):=a[j-1];     /* move the belt forward */
  for(j=0;j<=M-1;j=j+1){
     if(x[j]<S & b[j]>=0)        /* increment clocks of   */
      next(x[j]):= x[j]+1;       /* the busy stations     */
  }
}
```

and the processes $\texttt{lift\_drop}_j$ ($0 \leq j < \texttt{M}$) are defined as:

```
module lift_drop(a,b,x,j){

if(b[j] <= -2 & a[j+1] = -1){ /* idle station and new card*/
  next(b[j]):= 0..K;       /* generate a  personalization */
  next(a[j+1]):=b[j];      /* reset the slot              */
  next(x[j]):=0;           /* reset the clock             */
}
else if(b[j] >= 0 &  x[j] = S /* card personalized        */
      & a[j+1] = -2 ){        /* a blank slot beneath    */
  next(a[j+1]):=b[j];         /* drop the card            */
  next(b[j]):= -2;            /* reset the station        */
 }
}
```

Correctness

The desired correctness property is:

> *There exists a run that always produces personalized cards in the right order.*

To formalize the concept of "right order", an observer process is introduced that compares the output value with the expected value. Formally, the observer is defined as follows. We introduce a new state variable `out`, which initially is `0` and assume `K` is a multiple of `M`, say `2.M`. The behavior of the observer is specified by:

```
if(out = a[N-1])       next(out):= (out+1) mod K;
  else if(a[N-1]>-2)  next(out):= K;
```

If cards are not produced in the right order or if a card is output that has not been personalized, the observer sets the value of `out` to the "error" value `K`. The control objective then becomes to ensure that the observer will never detect an error. We can synthesize a scheduler that realizes this (if it exists) by asking SMV whether the following CTL formula holds:

$$\texttt{AF}\neg(\texttt{out} < \texttt{K}). \tag{6.1}$$

If this formula does *not* hold then there exists an infinite run in which for all states `out < K`, i.e., the observer never detects an error. In this case SMV will provide a counter example, which essentially is an infinite schedule for the machine that meets the control objective.

Optimization

Obviously, there are many runs in which all states satisfy `out < K`, for instance, a run in which the machine produces no cards at all. The interesting runs are those with high throughput, or more specifically with less number of blank slots in the output.

To minimize the blank slots in the output and in order to guide SMV towards optimal schedules, we introduce the "blank tolerance condition" of the machine, in the form of a new state variable `tl`, which is initially `0`, and is incremented and decremented as follows:

```
if(a[N-1]=-2)                                next(tl):=tl-1;
else if( a[N-1]>=0 & (a[N-1] mod S) = S-1)  next(tl):=tl+1;
```

We add `1` to `tl` each time `S` cards have been produced (a_{N-1} `modulo S = S-1`). We decrement `tl` with `1` whenever a blank slot arrives (`a`$_{N-1}$ `= -2`). However, we start decrementing only after the leading blank slots (`a[N-1] = -3`) have passed. In all other cases we leave the value of `tl` unchanged.

Now we ask SMV whether the following CTL formula holds:

$$\mathtt{AF}\neg(\mathtt{out} < \mathtt{K} \wedge \mathtt{tl} \geq \mathtt{0}). \tag{6.2}$$

If this formula does not hold, there exists an infinite scheduler that maintains the invariant $\mathtt{tl} \geq \mathtt{0}$. This means that each time when the system has produced `S` cards, the observer tolerates a single blank slot.

6.3 The Super Single Mode

Using the approach outlined in the previous section, the example run in Table 6.2 was generated. With a "normal-speed" PC we were able to generate example runs for $\mathtt{M} \leq \mathtt{5}$ (in the real machine `M` could be `8,16` or `32`). The runs exhibit the schedule of the super single mode as patented by Cybernetix. Table 6.2 shows the first 19 configurations of the the super single mode with `M = 4`, `S = 4`, `K = 12`. Each row represents a single configuration at a given time. The upper part of the row shows the values of the stations, while the lower part shows the values of the slots in the conveyor belt. An empty cell means the slot or the station is idle, a box (□) represents a new card, and a number represents the personalization value of the card contained in the station or in the slot. Table 6.2 can be read as:

- time 0: the machine is empty.
- time 1: first new card arrives on the conveyor belt.
- time 2: the first card is lifted to station `0`.
- time 4: the second card is lifted to station `1` and it continues likewise.
- time 5: there is no card from the input.
- time 6: station `0` finishes personalizing a card with value `0`. In super single mode, `M` (`4` in this example) time units are required to personalize a card.
- time 7: station `0` proceeds with personalizing another card with a different value (namely `4`). Note that value `3` is not taken yet. This pattern shows that the order of output is exactly the same as the order of the cards when they are fed into the machine, but the production order is different, and there is an overlap between rounds. This overlap is even more clearly visible when a machine with `8` (instead of `4`) personalization stations is considered.

If in our model a station is allowed to take more than `M` time units for personalizing a card, i.e., `S > M`, then CTL formula (6.2) holds. In other

Table 6.2: The super single mode for 4 personalization stations.

	in	stations				out
time	put	0	1	2	3	put
0						
1	□					
		0				
2	□					
		0				
3	□	□				
		0	1			
4	□	□				
		0	1			
5		□	□			
			1	2		
6	□	0	□			
		4	1	2		
7	□		0	□		
		4		2	3	
8	□	□	1	0		
		4	5	2	3	
9	□	□		1	0	
		4	5		3	
10		□	□	2	1	0
			5	6	3	
11	□	4	□		2	1
		8	5	6		
12	□		4	□	3	2
		8		6	7	
13	□	□	5	4		3
		8	9	6	7	
14	□	□		5	4	
		8	9		7	
15		□	□	6	5	4
			9	10	7	
16	□	8	□		6	5
		12	9	10		
17	□		8	□	7	6
		12		10	11	
18	□	□	9	8		7

words: if the conveyor belt is rolling faster than the personalization stations can handle then personalizing `M` consecutive cards becomes impossible.

Similarly, for a personalization time of `M` time units, if we have `M+1` consecutive new cards followed by empty slots (even with lots of empty slots), then it becomes impossible to personalize all of them. This result implies that the super single mode is optimal in the absence of errors.

6.4 Error Recovery

The control objective for the smart card personalization machine is to personalize cards in the right order even in the presence of errors. The super single mode, as explained above, only works for a perfect machine that makes no errors. In general, it is difficult to prevent errors from occurring (even though errors are rare, approximately 1 in 6000 cards), and so it makes our approach more realistic if we allow for the occurrence of errors in our model, and provide a means of recovering from them.

There are several methods to achieve fault-tolerant behavior. Our approach is inspired by the concept of *self-stabilization* [Dij74, Tel94], which is well-known from the area of distributed algorithms. An algorithm is called stabilizing if it eventually starts to behave correctly (i.e., according to the specification of the algorithm), regardless of the initial configuration.

Figure 6.3 shows the production cycle of the personalization machine under the super single mode. In the normal mode of operation the machine

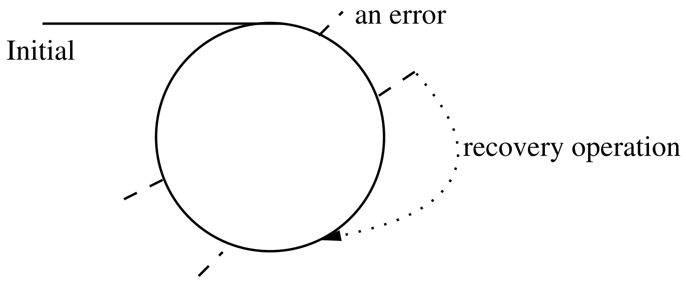

Figure 6.3: Stabilization of the smart card personalization system.

loops on the super single mode cycle (the continuous line). This loop is also shown in Table 6.2 with actual figures. The configurations of the machine at time 9, 10, 11, 12, 13 are equivalent (personalization value modulo `M = 4`) to the configurations at time 14, 15, 16, 17 and 18 respectively. Thus the super single mode enters the loop at time 9 and loops forever with a period of 5 time units.

However, when an error occurs (dashed line in figure 6.3), an error recovery treatment (dotted line) should be conducted to stabilize the system and bring it back to the loop. We use SMV to synthesize an error recovery treatment that brings the machine back to the loop. Basically, our approach is as follows:

1. Use SMV to synthesize a regular super single mode run, as described in the previous section.
2. Pick a state on this run and manually introduce an error; the new error state s now becomes the start state of the model.
3. Pick an arbitrary state t on the super single mode cycle, and encode this as an SMV state formula φ.
4. Ask SMV whether the following formula holds

$$\mathtt{AG}\neg\varphi. \tag{6.3}$$

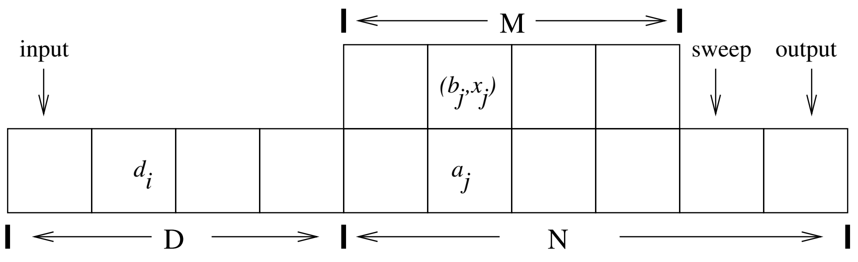

Figure 6.4: Expanded model of the smart card personalization machine.

> If formula (6.3) does not hold then SMV generates a counterexample; this counterexample is the schedule for a recovery operation that brings the system from state *s* back into super single mode.

Note that, unlike the theory of self-stabilization, we do not consider arbitrary initial configurations, but only configurations that have been obtained by introducing a single error into a super single mode configuration.

6.4.1 Types of Errors

It is easy to list many scenarios that can make the system behave erratically. In this paper we will only consider errors that may occur in the card. That is:

1. Type 1 errors (E1) are errors in a smart card originating from physical damage or other reasons. This type of error is detected by the personalization stations. In E1_a and E1_b in Table 6.3 are examples of E1 error.

2. Type 2 error (E2) are errors originating from the personalization station when cards are personalized wrongly, which makes them unusable. This type of error is detected by a tester situated at the end of the personalization stations. E2_a in Table 6.3 is an example of E2 error.

To make our system recoverable from these errors, we will modify our model in two ways: by adding extra operations and by expanding the belt in both directions.

6.4.2 Recovery Operations

If a defective card is detected in the tester then, in order to maintain correctness (i.e., produce personalized cards in the right order), the defective card has to be removed, a replacement card has to be produced, and inserted in the right position. In order to realize this, first the defective card

Table 6.3: The super single mode for 8 personalization stations with error. Only card values in station is shown

time	input	personalization stations								output
		0	1	2	3	4	5	6	7	(tester)
9										
10	□					4				
11	□	8								
12	□						5			
13	□		$E1_a$							
14	□							$E1_b$		
15	□			10?						
16	□								7?	
17	□				11?					
18										0
19	□					12?				1
20	□	16?								2
21	□						13?			$E2_a$
22	□		17?							4
23	□							14?		5
24	□			18?						$E1_b$
25	□								15?	7?
26	□				19?					
27										8

has to be swept off the belt, and then the belt has to go back to one of the personalization stations to retrieve a replacement card and place it in the right position. For these purpose we enrich our model with 'backward' and 'sweep' operations.

The backward move is the same as the forward move except that it moves the belt in the opposite direction. The forward move is the "normal" way of moving the belt, the backward move is used only to handle defective cards [Alb02]. We assume that a backward move takes 1 time unit per step.

When the belt moves backward, the leftmost cards on the belt are also pushed back to the edge. For technical reasons explained in [Alb02], the preferred way of treatment is to expand the belt to the left. As shown in Figure 6.4, the gap between the input station and the first personalization station, denoted by d_i ($0 \leq i \leq D$, $D = M$), is important for backward movement. Similarly, the belt is also expanded to the right: N (= M+2) covers the extended slots in the right side.

A sweeper is a device that kicks defective cards from the belt. In the

Table 6.4: Safety requirements for belt operations.

Operation	Safety requirements	meaning
backward	d_0 < 0	no processed card reaches input station, unprocessed (new) cards can return back to the input station
forward	a_{N-1} = out $\vee$ a_{N-1} = -2	no unexpected card reaches the tester station
sweep	a_M = K	only defective cards are swept

physical machine, a sweeper is situated after the personalization station. Formally the sweep operation is defined as:

```
module sweep(a){
  if(a[M]=K)     next(a[M]):=-2;
}
```

6.4.3 Safety Requirements

During the stabilization process, the machine executes operations that are not performed in super single mode. Even if the machine is allowed to perform these special operations, there are some safety requirements that have to be obeyed by the control program. These are shown in Table 6.4.

6.4.4 Results

For a single error scenario as defined above, there are `2.M` possible error configurations in one cycle of the super single mode. Using these error configurations as an initial state and the formula (6.3) we generated a recovery path that could stabilize the system back to the super single mode. Obviously, each path is different for different initial state, however, they share similar pattern. Thus we group similar paths together and explain their property below.

1. When the error type is E1 and the faulty card was detected in the first half stations (b_i: $0 \leq i \leq \lfloor \frac{M}{2} \rfloor$), then the faulty card remain in the station until a free slot is available. And the personalization value remains unused until next. For example When the faulty card E1$_a$ in Table 6.3 was detected the personalization value (which is 9) was used in station 2

2. Using the same technique, for E1 errors in the second half stations (b_i: $\lfloor \frac{M}{2} \rfloor < i \leq M-1$) will not solve the problem, instead it will introduce

Table 6.5: Defective card treatment for error type 2.

time	input	personalization stations								tester
		0	1	2	3	4	5	6	7	
12	□						5			
21	□						13			3
22	□		17							4
23	□							14		(E2)
24	□			5						6
25*	□	□		10	9	8	□	7	6	
26*	□		10	9	8	□	7	6		
32*	□	7	6	5						
38	□							14		5

another error. The generated recovery path for this scenario is to skip the personalization value for now and let the error evolve to E2 error. The personalization value (6) of $E1_b$ in Table 6.3 was skipped and $E1_b$ will be again an error of type E2 at time 24.

3. The recovery path for E2 errors consists:
   - finding a station with a fresh card, this station should be in the first half. Otherwise an error like $E1_b$ will happen again. See also Example 6.1.
   - rolling the belt backward to this station,
   - personalizing the card with the personalizations value which is missing, and
   - dropping the card to the belt and forward it to the tester.

Example 6.1 *In Table 6.5, at time* 23 *the* 5th *card is found defective. At the same time station* 6 *starts with a fresh card. If a replacement card would be produced in this station, then personalization number* 14 *would be skipped. But this will introduce another error, because the* 16th *and* 17th *cards are already in preparation and they can not be altered. Instead we can produce the card in the next station (station* 2*) that becomes available.*

6.4.5 Cost of Error Recovery

An upper bound on the number of time units spent recovering from an error can be calculated as follows.

1. Once an error is detected by the tester, one step forward may be necessary if it is an error like in Example 6.1.

2. To reproduce a replacement card we will require `S = M` time units, during this time the belt rolls back to the station.
3. Once the card is reproduced, it will take another `M` time units for the new card to reach the tester. In practice the belt can move forward faster than `M` time units, and the time spent to reach the tester will be smaller.

Thus, based on the above observation, `2.M + 1` time units are required in the worst case to recover from a single error. It is possible to tighten this upper bound by introducing fast forward and fast backward moves.

6.5 Conclusions

Using SMV, we rediscovered the super single mode that has previously been patented by Cybernetix. This result gives us new evidence that model checking can also be useful as a design aid for new machines. Our approach also allowed us to generate defective card treatments, that may arise due to damaged cards and wrong personalization. The present work shows error treatments for single error, we believe the same technique can be easily extended to multiple error treatment.

The input language of Cadence SMV is sufficiently expressive to encode in a natural and compact way a simplified model of the personalization machine. However, safety and liveness properties for multiple error treatments (of single or multiple types) are complicated to express in temporal logic, especially when dealing with the uncontrolled plant. Nevertheless, by decreasing the degree of uncontrollability of the plant, we believe multiple errors can be handled and more complex discrete time models of the actual Cybernetix design (including the controller) can be described.

A possible disadvantage of our approach is that the SMV descriptions are difficult to understand for people who are not familiar with formal methods (unlike say Petri nets). However, a clear advantage is that our description can serve directly as input for a powerful model checker.

Chapter 7

Analysis of the Zeroconf Protocol Using UPPAAL

with Frits W. Vaandrager and Miaomiao Zhang

Abstract We report on a case study in which the model checker UPPAAL is used to formally model parts of Zeroconf, a protocol for dynamic configuration of IPv4 link-local addresses that has been defined in RFC 3927 of the IETF. Our goal has been to construct a model that (a) is easy to understand by engineers, (b) comes as close as possible to the informal text (for each transition in the model there should be a corresponding piece of text in the RFC), and (c) may serve as a basis for formal verification. Our conclusion is that UPPAAL, which combines extended finite state machines, C-like syntax and concepts from timed automata theory, is able to model Zeroconf in a faithful and intuitive manner, using notations that are familiar to protocol engineers. Our modeling efforts revealed several errors (or at least ambiguities) in the RFC that no one else spotted before. We also identify a number of points where UPPAAL still can be improved. After applying a number of abstractions, UPPAAL is able to fully explore the state space of an instance of our model with three hosts.

7.1 Introduction

Our society increasingly depends on the correct functioning of modern communication technology. The most important and most often used protocols that describe the operation of this technology are standardized. It is surprising that protocols that are of such immense importance to our society are typically written in informal language, with frequent ambiguities, omissions and inconsistencies. They also fail to state what properties are expected of a network running the protocol, and what it means for an implementation to conform to a standard. By now there is ample evidence that formal (mathematical) techniques and tools may help to improve the quality of protocol standards (see e.g. [CGH$^+$93, BS98b, vLRG03, Sto03, Hol03, Rom04]). In order to avoid holes and ambiguities in standards the obvious way to go is to describe critical parts using programming and/or formal specification languages. There have been joint attempts of academia and industry to arrive at formal description languages for protocols. Interestingly — to the best of our knowledge — these languages have never been used in the authoritative part of protocol standards. Apparently, standardization bodies either did not trust/understand the formal specifications themselves or were afraid implementors would misinterpret them. Some protocol standard have extended finite state machines (EFSMs) inside, but these are mostly illustrative, not completely formal, and sometimes contain mistakes. Bruns and Staskauskas [BS98b] used C to describe the SONET Automatic Protection Switching (APS) protocol and report that developers found their C description easy to understand and superior to that which appeared in the APS standard. The lack of abstraction mechanisms is an obvious drawback of C.

The relationships between an (abstract) formal model of a protocol and the corresponding informal standard is typically obscure. As pointed out by [BM04], "current research seems to take the construction of verification models more or less for granted, although their development typically requires a coordinated integration of the experience, intuition and creativity of verification and domain experts. There is a great need for systematic methods for the construction of verification models to move on, and leave the current stage that can be characterized as that of *model hacking*. The adhoc construction of verification models obscures the relationship between models and the systems that they represent, and undermines the reliability and relevance of the verification results that are obtained."

In this chapter, we try to address the above problems and report on a case study where we use UPPAAL to formally model parts of Zeroconf, a protocol for dynamic configuration of IPv4 link-local addresses. Our goal has been to construct a model that (a) is easy to understand by engineers, (b) comes as close as possible to the informal text (for each transition in the model there should be a corresponding piece of text in the RFC), and (c) may serve as a basis for formal verification.

UPPAAL[BDL04], available at `www.uppaal.com`, is an integrated tool environment for formal specification, validation and verification of real time systems modeled as networks of timed automata [AD94]. The language for the new version UPPAAL 3.6 features a subset of the C programming language, a graphical user interface for specifying networks of EFSMs, and timed automata syntax for specifying timing constraints between events. Due to these extensions, the UPPAAL syntax appears to be sufficiently expressive for the description of critical parts of protocol specifications.

Zeroconf We describe and analyze (critical parts of) *Zeroconf* [CS05], a protocol for dynamic configuration of IPv4 link-local addresses that has been defined by the IETF Network Working Group in RFC 3927 [CAG05]. There are many situations in which one would like to use IP for local communication, for instance in the setting of in home digital networks or to establish communication between laptops. For these type of applications it is desirable to have a plug-and-play network in which new hosts automatically configure an IPv4 address, without using external configuration servers, like DHCP and DNS, or requiring users to set up each computer by hand. The Zeroconf protocol has been proposed by the IETF to achieve exactly this. It describes how a host may automatically configure an interface with an IPv4 address within the 169.254/16 prefix that is valid for communication with other devices connected to the same physical (or logical) link. The most widely adopted Zeroconf implementation is Bonjour from Apple Computer, but several other implementations are available.

Contribution The contribution of this chapter is, first of all, a formal model of a critical part of Zeroconf — a protocol with clear practical relevance — that is easy to understand, faithful to the RFC, and with an extensive discussion of the relationship between the model and the RFC. Our efforts revealed several errors (or at least ambiguities) in the RFC that no one else spotted before. We also identify several directions where UPPAAL still can be improved. Finally, after applying several abstractions we manage to establish some key correctness properties of an instance of our model with three hosts.

Related Work Zeroconf involves a number of probabilistic aspects that are not incorporated in our UPPAAL model: hosts select IP-addresses randomly using a pseudo-random number generator, and at some point during the protocol they wait for a random amount of time selected uniformly from an interval. The probabilistic behavior of Zeroconf has been studied in [BSHV03, KNPS03]. The primary goal of [BSHV03] was to investigate the trade off between reliability and effectiveness of the protocol using a stochastic cost model. The model of [BSHV03], which only involves a single host, is quite appropriate in capturing the probabilistic behavior of IP address configuration and conflict handling, but the analysis takes place at a level that is much more abstract than the RFC. Based on an earlier version of [GVZ06b], a more detailed model has been presented in [KNPS03] using the probabilistic model checker PRISM [KNP04]. The model checking results reported in [KNPS03] are very interesting, but the precise relationship between the model and the RFC is unclear (for instance, in the model of [KNPS03] address defense only occurs *before* a host is using an IP address). Our motivation for using UPPAAL instead of PRISM was that the input language of PRISM is too primitive for our purposes (no GUI, just a few datatypes, no support of C-like syntax,..). A toolset that combines the functionality of UPPAAL and PRISM would be ideal for dealing with the Zeroconf protocol.

7.2 The Protocol

We now describe the Zeroconf protocol, our UPPAAL model of it, and the relationship between our model and RFC 3927 [CAG05], the official protocol standard.

A Zeroconf network is composed of a set of hosts on the same link. Hosts in the Zeroconf network can be devices that are present at home, office, embedded systems "plugged together" as in an automobile, or the laptops of three friends who are writing a joint paper and want to share a file. The goal of Zeroconf is to enable networking in the absence of configuration and administration services. The core of RFC 3927 [CAG05] concerns the dynamic configuration of IPv4 link-local addresses, and this is the part on which we will focus in this chapter.

The basic idea of Zeroconf is trivial and easy to explain. A host that wants to configure a new IP link-local address randomly selects an address from a specified range and then broadcasts a few identical messages to the other hosts, seperated by some delay, asking whether someone is already using the address. If one of the other hosts indicates that it is using the other address, the host starts all over again. Otherwise, it may start using the address after waiting a certain amount of time.

One may view Zeroconf as a distributed mutual exclusion algorithm in which the resources are IP addresses. A goal of Zeroconf is to prevent that at any point two different hosts are using the same IP address. The underlying algorithm used in Zeroconf is similar to Fisher's mutual exclusion algorithm [AL94] and makes essential use of timing. However, whereas Fischer's algorithm uses a shared variable for communication between processes, Zeroconf uses broadcast communication. Within Zeroconf, hosts do not aim at acquiring access to a specific critical section (IP address); it is enough to obtain access to one of the 65024 available critical sections (IP addresses).

7.2.1 Basic Modelling Assumptions

RFC 3927 assumes a *set* of hosts. This set is not fixed and host may join and leave while the protocol is running. Since UPPAAL does not support dynamic process creation, we assume a fixed number of `k` hosts. It may take arbitrary long before a host becomes active in the protocol and one may argue that in this way creation of new hosts is being captured. We do not model host failure or termination but it would be easy to add this. In our model, a host that has configured an IP address may stop sending messages. From an observational point of view this is the same as a (stopping) failure. A phenomenon that may occur in practice, and which we have also not modeled here, is that previously separate Zeroconf networks are joined.

The behavior of each host is modeled by three timed automata that are composed in parallel: `Config`, `InputHandler` and `Regular`. Automaton `Config` models the configuration of a new IP address, `InputHandler` takes care of the incoming messages, and `Regular` is an abstract model of the activity of all the other processes running on the host. All three automata are parametrized by the hardware address of the host they belong to. For convenience, in our model a hardware address is a natural number in the range `0` to $\mathtt{k}-1$. Within UPPAAL, the scalarset type `scalar[k]` denotes the set $\{0,\ldots,\mathtt{k}-1\}$: `typedef scalar[k] HAType`

On scalarsets, only restricted operations are permitted. As a consequence, a scalarset is a fully symmetric type and the behavior of a model is invariant under arbitrary permutations of the elements of a scalarset [ID93, HBL$^+$03]. By defining a scalarset type rather than a subrange, we tell UPPAAL that all the hardware addresses (and therefore also the hosts) play a fully symmetric role, which makes it possible to exploit this symmetry during exploration of the state space.

7.2.2 The Network

RFC 3927 states the following assumption about the underlying network [page 4, section 1.3]:

> "This specification applies to all IEEE 802 Local Area Networks (LANs) [802], including Ethernet [802.3], Token-Ring [802.5] and IEEE 802.11 wireless LANs [802.11], as well as to other link-layer technologies that operate at data rates of at least 1 Mbps, have a round-trip latency of at most one second, and support ARP [RFC826]."

The Address Resolution Protocol (ARP [Plu82]) is widely used method for converting protocol addresses (e.g., IP addresses) to local network ("hardware") addresses (e.g., Ethernet addresses). It allows dynamic distribution of the information needed to build tables to translate protocol addresses to hardware addresses. Within Zeroconf all messages are ARP packets. For our model, the relevant information in an ARP packet consists of (1) a sender hardware address, (2) a sender IP address, (3) a target IP address, and (4) the type of the packet, which can be either "request" or "reply". Hence, an ARP packet can be defined as a UPPAAL C data type as follows:

```
typedef struct{
        HAType   senderHA;  // sender hardware address
        IPType   senderIP;  // sender IP address
        IPType   targetIP;  // target IP address
        bool     request;   // is the packet a Request or a Reply
}ARP_packet;
```

Here we use the convention that the `request` field is `true` for ARP requests and `false` for ARP replies. A host that is looking for the local network address of another host with IP address x, broadcasts an ARP request packet with the field `targetIP` set to x. A host with IP address x will then return an ARP reply packet with the field `senderHA` set to its local network address.

In Zeroconf, all ARP packets are broadcast [page 13, section 2.5]:

> "All ARP packets (*replies* as well as requests) that contain a Link- Local 'sender IP address' MUST be sent using link-layer broadcast instead of link-layer unicast. This aids timely detection of duplicate addresses."

We model the underlying network as a set of `n` identical `Network` automata. Each of these automata takes care of handling a single ARP request at a time. To express that all the automata are symmetric, we define a scalar set `NetworkType` and parametrize each automaton by an element `j` from this type.

The main reason for having `n` automata is that this allows us to model round-trip latencies in UPPAAL. Fig. 7.1 schematically illustrates the operation of a `Network` automaton. After a request from a host comes

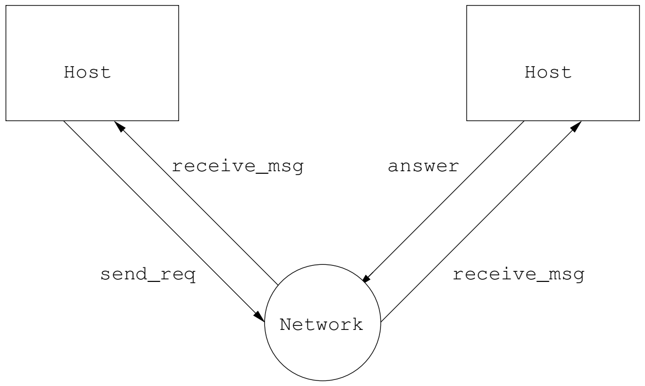

Figure 7.1: Interaction between `Network` automaton and hosts.

in (`send_req`), this is broadcast by a `Network` automaton to all hosts (`receive_msg`). In case there is a corresponding answer (this may be a reply or a request packet) this is accepted (`answer`) and also broadcast to all hosts (`receive_msg`). All these interactions take place within 1 second. After completing its task the `Network` automaton returns to its initial location, ready to take care of a new request.

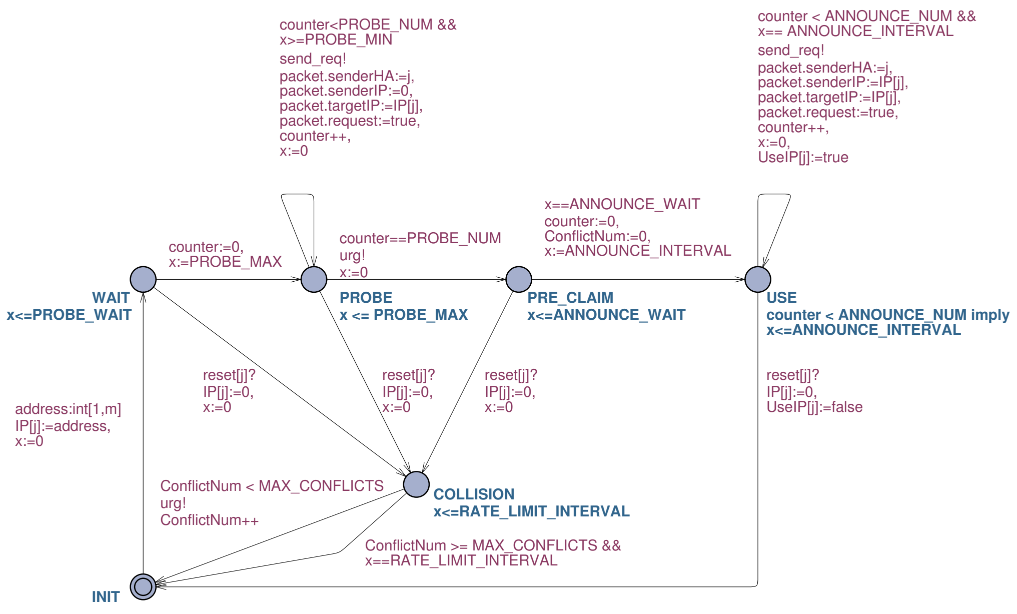

Figure 7.2: Automaton `Config`.

To simplify our model, we assume that a host handles an incoming ARP request in zero time, i.e., we adopt the synchrony hypothesis that is well-known from synchronous programming [BG92].

Before explaining our UPPAAL model of the `Network` automaton in detail (in Section 7.2.6), we now turn our attention to the core part of RFC 3927, which concerns address configuration.

7.2.3 Address Configuration

Fig. 7.2 displays the automaton `Config[j]`, which specifies how host `j` configures a new IP address.

Each host starts in location `INIT`, where it resides until it has selected an IP address. According to the RFC [page 9, section 2.1]:

> "When a host wishes to configure an IPv4 Link-Local address, it selects an address using a pseudo-random number generator with a uniform distribution in the range from 169.254.1.0 to 169.254.254.255 inclusive. The IPv4 prefix 169.254/16 is registered with the IANA for this purpose. The first 256 and last 256 addresses in the 169.254/16 prefix are reserved for future use and MUST NOT be selected by a host using this dynamic configuration mechanism."

Just to keep the code simple, we abstract sligthly from the naming of IP addresses. An IP address simply is a number in the range `0` to `m`, where `m` denotes the number of available link-local addresses: The address `0` corresponds to the all zeroes IP address 0.0.0.0, which is used as a special 'unknown' or 'undefined' value in the protocol, and the addresses `1` to `m` correspond to the addresses registered with the IANA, listed in increasing order. Due to the special role of the address `0`, we cannot declare `IPType` as a (fully symmetric) scalarset, and thus we declare it as a subrange instead. A transition from location `INIT` to location `WAIT` takes place when an address has been selected. Via the UPPAAL select statement `address:int[1,m]`, we nondeterministically bind identifier `address` to a value in the interval `[1,m]`. This means that there is an instance of the transition for each number in this interval. In this way, we express that an IP address is chosen nondeterministically. The selected address is stored in state variable `IP[j]`. The RFC continues [page 11, section 2.2.1]:

> "When ready to begin probing, the host should then wait for a random time interval selected uniformly in the range zero to PROBE_WAIT seconds, and should then send PROBE_NUM probe packets, each of these probe packets spaced randomly, PROBE_MIN to PROBE_MAX seconds apart."

The waiting period is modeled by resetting a local clock `x` upon entering location `WAIT` and by bounding the time the host may stay in `WAIT` with an invariant `x <= PROBE_WAIT`. At any point the host may move to location `PROBE`, where it starts sending "probes". The notion of an ARP Probe is specified in the RFC as follows:

> "A host probes to see if an address is already in use by broadcasting an ARP Request for the desired address. The client MUST fill in the 'sender hardware address' field of the ARP Request with the hardware address of the interface through which it is sending the packet. The 'sender IP address' field MUST be set to all zeroes, to avoid polluting ARP caches in other hosts on the same link in the case where the address turns out to be already in use by another host. The 'target hardware address' field is ignored and SHOULD be set to all zeroes. The 'target IP address' field MUST be set to the address being probed. An ARP Request constructed this way with an all-zero 'sender IP address' is referred to as an "ARP Probe"."

Sending ARP Probes is modeled via actions `send_req[j]!` that synchronize with the network. The actual packet is communicated via a global shared variable `packet` of type `ARP_packet`: in UPPAAL the assignments in an output (!) transition are executed before the assignments in a synchronizing input (?) transition, and this allows us to assign a value to `packet` in a `send_req[j]!` transition, which is then picked up by a corresponding `send_req[j]?` transition by a `Network` automaton. The lower and upper bounds of the probe interval are expressed in our model with a guard `x`

`>= PROBE_MIN` on the sending transition and an invariant `x <= PROBE_MAX` on location `PROBE`, respectively. By setting `x` to `PROBE_MAX` in the transition from `WAIT` to `PROBE`, we express that the first probe is sent immediately. A local variable `counter` is used to record the number of probes that have been sent. After the probing phase is successfully completed, the automaton jumps to location `PRE_CLAIM`. The urgent broadcast channel `urg` ensures that this transition is taken as soon as it is enabled. As the reader can check, the translation from the RFC description of the probing phase to UPPAAL is straightforward.

According to the RFC:

> "If, by ANNOUNCE_WAIT seconds after the transmission of the last ARP Probe no conflicting ARP Reply or ARP Probe has been received, then the host has successfully claimed the desired IPv4 Link-Local address."

Clock `x` is used to ensure that exactly `ANNOUNCE_WAIT` time units are spent in location `PRE_CLAIM`. A transition from location `PRE_CLAIM` to location `USE` is taken to indicate that the host has successfully claimed an address.

In our model, automaton `InputHandler[j]` (which will be explained in Section 7.2.4) takes care of handling incoming messages. If `InputHandler[j]` decides that, due to some conflict, a new address must be configured, it sends a `reset[j]` signal to automaton `Config[j]`. Upon receiving this signal, `Config[j]` sets `IP[j]` to 0 and jumps to location `COLLISION`. According to the RFC:

> "A host should maintain a counter of the number of address conflicts it has experienced in the process of trying to acquire an address, and if the number of conflicts exceeds MAX_CONFLICTS then the host MUST limit the rate at which it probes for new addresses to no more than one new address per RATE_LIMIT_INTERVAL. This is to prevent catastrophic ARP storms in pathological failure cases, such as a rogue host that answers all ARP Probes, causing legitimate hosts to go into an infinite loop attempting to select a usable address."

A counter `ConflictNum` is used in our model to record the number of conflicts that have occurred during the process of acquiring an IP address. Depending on the value of `ConflictNum`, the automaton returns to location `INIT` immediately or first waits for `RATE_LIMIT_INTERVAL` time units. Again, the correspondence between the RFC text and our UPPAAL model is straightforward.

In location `USE` the host announces the new address that it has just claimed [page 12, section 2.4]:

> "Having probed to determine a unique address to use, the host MUST then announce its claimed address by broadcasting

as many as ANNOUNCE_NUM ARP announcements, spaced ANNOUNCE_INTERVAL seconds apart. An ARP announcement is identical to the ARP Probe described above, except that now the sender and target IP addresses are both set to the host's newly selected IPv4 address. The purpose of these ARP announcements is to make sure that other hosts on the link do not have stale ARP cache entries left over from some other host that may previously have been using the same address."

The above description is ambiguous/incomplete at 3 points. First of all, the RFC does not specify upper and lower bounds on the time that may elapse between sending the last ARP Probe and sending the first ARP Announcement. However, according to the protocol designers upper and lower bound both equal `ANNOUNCE_WAIT` [Che06]. Also, the RFC does not specify whether a host may immediately start using a newly claimed address (in parallel with sending the ARP Announcements), or whether it should first send out all announcements. According to the designers, a host should send the first ARP Announcement, and then it can immediately start using the address [Che06]. So the second announcement goes out `ANNOUNCE_INTERVAL` seconds later, but other traffic does not need to be held up waiting for that. Finally, the RFC does not specify the tolerance that is permitted on the timing of ARP Announcements. Since no physical device can consistently send messages spaced *exactly* `ANNOUNCE_INTERVAL` seconds apart, strictly speaking it is impossible for an implementation to conform to the RFC. According to the designers, the RFC does not specify accuracy requirements, partly because the protocol is robust to a wide range of variations, so it does not matter [Che06]. We decided to follow the RFC and not specify accuracy requirements, but if someone wants to use our model for automatic generation of tests, for instance using the UPPAAL-TRON toolset [LMN05], he or she will have to modify our model at this point.

With this additional information, the modeling of the announcement phase in UPPAAL is straightforward and analogous to that of the probing phase. After sending the first announcement, Boolean variable `UseIP[j]` is set to `true`. This enables an automaton `Regular[j]`, to start sending out regular ARP request packets with the `senderIP` field set to `IP[j]` and the `targetIP` field set to an arbitrary link-local address. However, even when a host is using an IP address still at any moment a conflict may arise. When this happens automaton `Config[j]` returns to its initial location and `UseIP[j]` is set to `false` again.

7.2.4 Input Handler

Automaton `InputHandler[j]` receives incoming ARP packets and decides what to do with them. Input handling is described at various places in RFC 3937, which makes it nontrivial to determine the reaction to an ar-

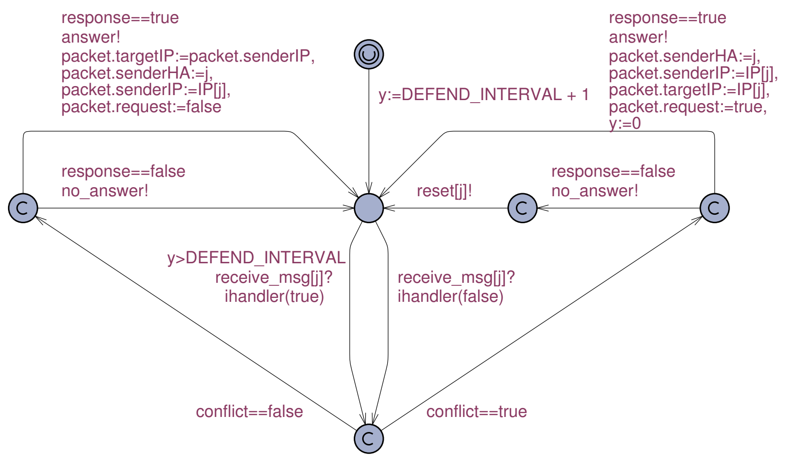

Figure 7.3: Automaton `InputHandler[j]`.

bitrary ARP packet, also because Zeroconf runs on top of the ARP protocol, which it sometimes follows but sometimes overrules. Automaton `InputHandler` is displayed in Fig. 7.3. When a new packet arrives, that is, when a `receive_msg[j]?` transition occurs, the automaton calls a function `ihandler` to find out what to do. This function computes two bits, `conflict` and `response`: if `conflict==true` then some other host is using or trying to use the IP address the host has selected and if `response==true` then a packet will be send in response. Thus the value of the two bits determines the reaction of the input handler to the incoming packet:

1. If `conflict==true` and `response==false`, a `reset[j]` signal is sent.
2. If `conflict==true` and `response==true`, an ARP Announcement is broadcast.
3. If `conflict==false` and `response==true`, an ARP Reply is broadcast.
4. If `conflict==false` and `response==false`, the packet is ignored.

Clock `y` is used to measure the time since the last conflict. The definition of `ihandler` is listed in Fig. 7.4. Function `ihandler` has a parameter `defend` which may be either `false` or `true`. This parameter, which indicates that a host will defend its IP address in case of a conflicting ARP request, may be `true` only if there has been no other conflict during the last `DEFEND_INTERVAL` time units. Altogether, the input handler has to distinguish 9 scenarios (A)-(I). These scenarios are described in detail below. The systematic classification of these scenarios revealed two more ambiguities/mistakes in the standard.

```
void ihandler(bool defend) {
  if (IP[j]==0) // Scenario A: I have not selected an IP address
        {response:=false; conflict:=false;}
  else if (packet.senderHA==j) // Scenario B: I have sent the packet myself
        {response:=false; conflict:=false;}
  //There is a conflict: somebody else is using my IP address!
  else if (packet.senderIP==IP[j])
        { conflict:=true;
        if (not UseIP[j]) // Scenario C: select a new address
              response:=false;
        else if (defend) // Scenario D: I am going to defend my address
              response:=true;
        else  // Scenario E: I will not defend my address
              response:=false; }
  else if (not UseIP[j])
        { response:=false;
    // Scenario F: conflicting probe
    if (packet.targetIP==IP[j] && packet.request && packet.senderIP==0)
                 conflict:=true;
    else  //Scenario G: Packet is not conflicting with IP address that I want to use
             conflict:=false; }
  else // Incoming packet is not conflicting with IP address that I am using
        { conflict:=false;
        // Scenario H: answer regular ARP request
        if (packet.targetIP==IP[j] && packet.request)
             response:=true;
        else // Scenario I: no reply message required
             response:=false; } }
```

Figure 7.4: Function `ihandler`.

7.2.5 Scenarios for Input Handler

Scenario A Clearly, if a packet comes in when a host has not yet selected an IP address it should be ignored. This scenario is not listed explicitly in the RFC but should be obvious.

Scenario B Packets that a host has sent itself can be ignored. Also this scenario is implicit in the RFC.

Scenario C A conflict may arise when another host sends a packet with the `senderIP` field set to `IP[j]`. This occurs in Scenario C, which is described on [page 11, section 2.2.1]:

> "If during this period, from the beginning of the probing process until ANNOUNCE_WAIT seconds after the last probe packet is sent, the host receives any ARP packet (Request *or* Reply) on the interface where the probe is being performed where the packet's 'sender IP address' is the address being probed for, then the host MUST treat this address as being in use by some other host, and MUST select a new pseudo-random address and repeat the process."

Scenarios D and E In the previous scenario, `UseIP[j]==false`. The case with `UseIP[j]==true` is also described in the RFC [page 12, section 2.5]:

> "Address conflict detection is not limited to the address selection phase, when a host is sending ARP Probes. Address conflict detection is an ongoing process that is in effect for as long as a host is using an IPv4 Link-Local address. At any time, if a host receives an ARP packet (request *or* reply) on an interface where the 'sender IP address' is the IP address the host has configured for that interface, but the 'sender hardware address' does not match the hardware address of that interface, then this is a conflicting ARP packet, indicating an address conflict.
> A host MUST respond to a conflicting ARP packet as described in either (a) or (b) below:
> (a) Upon receiving a conflicting ARP packet, a host MAY elect to immediately configure a new IPv4 Link-Local address as described above, or
> (b) If a host currently has active TCP connections or other reasons to prefer to keep the same IPv4 address, and it has not seen any other conflicting ARP packets within the last DEFEND_INTERVAL seconds, then it MAY elect to attempt to defend its address by recording the time that the conflicting ARP packet was received, and then broadcasting one single ARP Announcement, giving its own IP and hardware addresses as the sender addresses of the ARP. Having done this, the host can then continue to use the address normally without any further special action. However, if this is not the first conflicting ARP packet the host has seen, and the time recorded for the previous conflicting ARP packet is recent, within DEFEND_INTERVAL seconds, then the host MUST immediately cease using this address and configure a new IPv4 Link-Local address as described above. This is necessary to ensure that two hosts do not get stuck in an endless loop with both hosts trying to defend the same address.
> A host MUST respond to conflicting ARP packets as described in either (a) or (b) above. A host MUST NOT ignore conflicting ARP packets."

Case (a) corresponds to our scenario E. This scenario occurs when the right `receive_msg?` transition in the automaton is taken, which sets `defend` to `false`, Case (b) corresponds to scenario D. This scenario occurs when the left `receive_msg?` transition is taken, which sets `defend` to `true`.

The interpretation of "and it has not seen any other conflicting ARP packets within the last DEFEND_INTERVAL seconds" in the previous quotation from the RFC is not clear. Is a host allowed to defend its address if there has been a recent conflict concerning a *different* address (but no previous conflict concerning the current address)? Strictly speaking, the host has seen a conflicting packet and it may not defend. However, the conflict concerned a different address, and the motivation for recording the time since the last conflict has been to rule out a scenario in which two hosts get stuck in an endless loop trying to defend the *same* addess. Thus one could also argue that in this situation a host may defend its address.

To model this interpretation, one would have to add an assignment `y := DEFEND_INTERVAL+1` to the reset transition of the input handler.

Scenarios F and G The RFC specifies one more conflict scenario [page 11, section 2.2.1]:

> "In addition, if during this period [from the beginning of the probing process until ANNOUNCE_WAIT seconds after the last probe packet is sent] the host receives any ARP Probe where the packet's 'target IP address' is the address being probed for, and the packet's 'sender hardware address' is not the hardware address of the interface the host is attempting to configure, then the host MUST similarly treat this as an address conflict and select a new address as above. This can occur if two (or more) hosts attempt to configure the same IPv4 Link-Local address at the same time."

In the `ihandler` code, this corresponds to scenario F. Scenario G, which is implicit in the RFC, occurs when the incoming packet is not conflicting and the host is not yet using an IP address. In this case the incoming packet is ignored.

Scenario H and I The Address Resolution Protocol (RFC 826) [Plu82] specifies that if a host receives an ARP request packet, it should return an ARP reply packet if it uses an IP address that equals the target protocol address of this request. In the reply packet the hardware and protocol field should be swapped, putting the local hardware and protocol addresses in the sender fields. Zeroconf (RFC 3927) is not explicit about conformance to RFC 826, but in our model we take the view that once a host is using an IP address, it answers regular ARP requests in agreement with RFC 826 except when (a) the request has been broadcast by the host itself, or (b) there is a conflict. This is scenario H in our model. The final Scenario I occurs when the incoming packet is not conflicting with the IP address that the host is using, and no reply packet needs to be sent.

Note that in automaton `InputHandler[j]` some of the locations are committed (C). In UPPAAL, when a system reaches a committed location, the next transition has to be an outgoing transition from that location. The

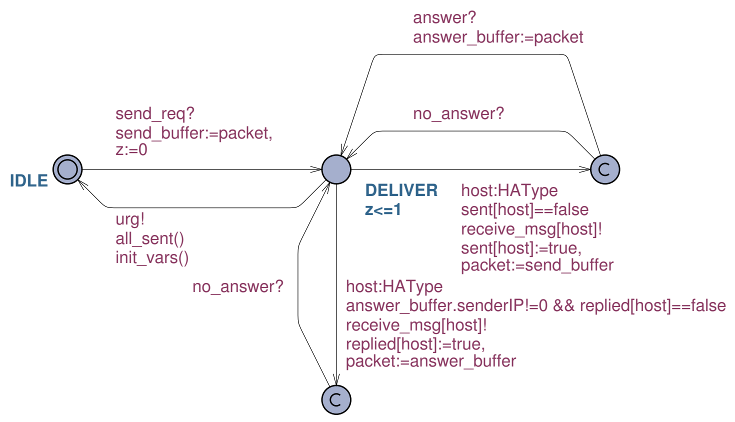

Figure 7.5: The `Network` automaton.

use of committed locations here is a modeling trick. When a network automaton delivers a packet to an input handler via a `receive_msg` synchronization, the input handler has to return an answer (if there is one) instantaneously (by the synchrony hypothesis). But since in general there are many network automata active, we need to ensure that the answer is picked up by the right automaton. Introducing separate channel names for each network automaton or pi-calculus like private channels would create too much overhead. Our trick is that a network automaton may only synchronize on an `answer` action right after performing a `receive_msg` action. By making the locations of the input handler following a `receive_msg` transition committed, we ensure that the reply is picked up by the right network automaton. Essentially, the `receive_msg` and `answer` synchronizations take place in a single atomic transaction. In case the input handler does not generate an answer, it uses a `no_answer` action to inform the network automaton about this. This synchronization is an artifact of our model since in reality no signal is sent.

7.2.6 The Network Automaton

The `Network` automaton is shown in Fig. 7.5. Initially the automaton is in its `IDLE` location. As soon as it receives a packet from a host via `send_req`, it jumps to the `DELIVER` location. Since there is no lower bound on message delivery time, message delivery may start immediately. A local clock `z` is reset to zero and an invariant $z \leq 1$ ensures that within 1 second the network broadcasts the packet (and the answer if there is one) to all hosts. In our model we assume that there is at most one host that wants to answer any given request, and that an answer does not induce subsequent answers. It is possible to modify the `Network` automaton so that it can handle multiple and successive answers, but this requires additional state variables and more complicated data structures. Our `Network` automaton has two local buffers: `send_buffer` stores the packet that was sent by the host and `answer_buffer` stores an answer when it arrives. In addition, `Network` maintains Boolean arrays `sent` and `replied` to record to which hosts the packets have already been delivered. Using the UPPAAL select statement, the automaton non deterministically selects in which order a packet is delivered to the different hosts. A host may return an answer upon receipt of a request, as explained in Subsection 7.2.4. The lower transition labeled with `receive_msg` is enabled as soon as there is an answer packet in `answer_buffer`. The network returns to its `IDLE` location and resets its buffers, as soon as all messages have been sent. This is checked by the Boolean function `all_sent`. Upon return to the `IDLE` location all variables are re-initialized.

7.2.7 Dimensioning the Complete Model

The RFC [page 25, section 9] specifies the following values for the different timing constants. These definitions are copied almost verbatim in the UPPAAL declaration section of our model.

```
"PROBE_WAIT          1 sec (initial random delay)
PROBE_NUM           3     (number of probe packets)
PROBE_MIN           1 sec (minimum delay till repeated probe)
PROBE_MAX           2 sec (maximum delay till repeated probe)
ANNOUNCE_WAIT       2 sec (delay before announcing)
ANNOUNCE_NUM        2     (number of announcement packets)
ANNOUNCE_INTERVAL   2 sec (time between announcement packets)
MAX_CONFLICTS      10     (max conflicts before rate limiting)
RATE_LIMIT_INTERVAL 60 sec (delay between successive attempts)
DEFEND_INTERVAL    10 sec (minimum time between defensive ARPs)."
```

A Zeroconf network has 65024 IP addresses available and it is suitable for up to 1300 hosts [CAG05]. These values are too big for automatic verification and with 3 hosts and 65024 IP addresses also the UPPAAL simulator runs out of memory. A next issue regarding the dimensioning of the model is the number `n` of `Network` automata, i.e., the maximal number of ARP requests that may be in transit at any given point. In our model, a host may select an IP address, send a probe, and return to the initial location via a reset in zero time. In fact, this behavior may be repeated `MAX_CONFLICTS` times in a row in zero time. Once a host is using an IP address, the number of messages in transit may increase even further (in fact unboundedly) since there is no lower bound on the time between successive ARP requests. UPPAAL forces us to bound the number of `Network` automata to some number `n`.

7.3 Verification

The model described in Section 7.2 is very close to the RFC definition of the protocol. However, it is too big for UPPAAL to do a complete state space exploration for nontrivial instances, even when we use symmetry reduction.

The RFC does not specify what properties the protocol must satisfy. However, it is clear that at least the following two correctness properties are desirable:[1]

1. Mutual exclusion, i.e., no two hosts may use same IP address:
```
ME = A[] forall (i: HAType) forall (j: HAType)
        (UseIP[i] && UseIP[j] && IP[i]==IP[j])
                                     imply i==j.
```
2. The network has no deadlock, i.e, in each reachable state a transition is possible: `DL = A[] not deadlock`.

Using the latest version of UPPAAL (3.6 beta), we only managed to establish `ME` and `DL` for the instance with 2 hosts, 1 IP address and 2 network automata. Nevertheless, it is rather obvious that Zeroconf satisfies the mutual exclusion property and is free of deadlocks. In the remainder of this section, we first discuss a manual proof of mutual exclusion and then outline an abstracted version of our model that can be fully explored by UPPAAL in the case of 3 hosts and used to prove mutual exclusion automatically for this instance. We claim that the full model has no deadlocks but do not present the (long and tedious) proof here. Since the abstract model overapproximates the full model, absence of deadlock in the first does not imply absence of deadlock in the second.

In the full version of our paper, we present a short, manual, operational proof of the mutual exclusion property for the general model. Inspection of the proof indicates that Zeroconf is extremly robust: the protocol has been designed to handle all kinds of error scenarios (loss of messages, failure of hosts, merge of networks) which do not occur within our idealized model. Without these errors, it suffices (for mutual exclusion) to send out a single probe (`PROBE_NUM=1`), there is no need for sending announcements (`ANNOUNCE_NUM=0`), and a host may start using an address after waiting any time longer than the maximal communication delay. For a model of this simplified protocol with 3 hosts UPPAAL can verify `ME` and `DL` in a few seconds on a standard PC.

To make automatic verification of mutual exclusion possible for the full protocol in the case with 3 hosts, we had to apply a combination of sev-

[1]Mutual exclusion will not hold in an extension of our model in which Zeroconf networks can be merged. In such an extension the specification should be weakened: mutual exclusion may be violated after a join, but as soon as the violation is detected (due to an ARP packet) mutual exclusion will be restored within a specified amount of time (provided meanwhile no further joins occur).

eral abstractions (on top of the abstractions that are already applied by UPPAAL): dead variable reduction, as it has been studied in the PhD thesis of Yorav [Yor00], and also overapproximation by weakening guards or by making an urgent channel non-urgent. We refer to the full version of this paper for details. Also, we had to make the additional assumption that at any time for each host there is at most one outgoing message in transit. This allows us to associate a single network automaton to each host, which only accepts packets from this host when empty. Using the combination of the above abstractions, we were able to prove mutual exclusion for instances of Zeroconf with 2 hosts and up to 5 IP addresses, and an instance with 3 hosts and 1 IP address.

We also did some experiments with the use of symmetry reduction for IP addresses. Since in Zeroconf the IP address 0 (i.e., 0.0.0.0) plays a special role, and UPPAAL can only handle fully symmetric data types, this required some rewriting of the model. Using symmetry reduction for IP addresses, we were able to establish mutual exclusion for a system with 2 hosts and an *arbitrary* number of IP addresses. Essentially, this is due to a theorem of Ip and Dill [ID93] on *data saturation*. This theorem (which was proved in the setting of Murphi but can easily be shown to carry over to UPPAAL) states that for certain ("data") scalarsets, the state graph does not grow any further once the size of the scalarsets grows beyond the number of scalarset locations in the system. In the case of 2 hosts, the number of scalarset locations for IP addresses in the model equals 12 (1 for each `Config[j]` automaton, 4 for each `Network` automaton, and 2 for the `packet` variable). In fact, data saturation already happens starting from scalarsets of size 5. Actually, we conjecture that there exists a bisimulation between a model with n IP addresses, for any n, and the model with just one (nonzero) IP address, via which a proof of `ME` for the general model can be reduced to a proof of `ME` for the model with just one address.

7.4 Conclusions

Our goal has been to construct a model of Zeroconf that (a) is easy to understand by engineers, (b) comes as close as possible to RFC 3927, and (c) may serve as a basis for formal verification. Did we succeed?

Understandability Of course, it is not to us to judge whether our model is understandable for others. The present chapter aims to place the cards on the table as a basis for a discussion. The UPPAAL syntax, which combines extended finite state machines, C-like syntax and concepts from timed automata, will certainly be familiar to protocol engineers, except maybe for the use of clock variables. However, our experience is that timed automata notation is easy to explain, also to people without expertise in theoretical computer science. Clocks provide a simple and intuitive means to specify the various timing constraints in Zeroconf. The automata `Config` and `InputHandler` would be the obvious candidates for inclusion in a standard. The only elements in these automata which may be considered less intuitive are the use of committed locations in the `InputHandler` and the sending of a `no_reply` signal in situations where no reply packet is sent (this is an artifact of the model since in reality there is no such signal). However, we can easily remove these elements from the `InputHandler` automaton at the price of making the `Network` automata (somewhat) more complicated.

There are at least four extensions of the UPPAAL syntax that would help us to further improve the readability of our model: (1) A richer syntax for datatypes, for instance permitting us to write 0.0.0.0 for the all zero IP address instead of 0. (2) The ability to initialize clock variables, allowing us to eliminate the initial transition in the `InputHandler[j]` automaton. (3) The ability to test clocks within the body of functions, allowing us to move the test on `y` into the definition of `ihandler`, where it belongs conceptually. (4) Urgent transitions as advocated in [GV05]. This would allow us e.g., to replace the invariant

```
counter < ANNOUNCE_NUM imply x <= ANNOUNCE_INTERVAL
```

in automaton `Config` by an urgency predicate

```
x <= ANNOUNCE_INTERVAL.
```

In our opinion urgency predicates are more intuitive than location invariants. Once these extensions have been implemented, a good case can be made for inclusion of the `Config` and `InputHandler` automata (with the `ihandler` code) in a Zeroconf standard. These models definitely help to clarify the RFC and to prevent incorrect interpretations due to ambiguity in the textual part. The UPPAAL simulator is also very useful to obtain insight in the protocol.

Our efforts revealed five places where RFC 3927 [CAG05] is incomplete/unclear:

1. No upper and lower bounds are given on the time that may elapse between sending the last ARP Probe and sending the first ARP Announcement.
2. It is not specified whether a host may immediately start using a newly claimed address or whether it should first send out all ARP Announcements.
3. No tolerance is specified on the timing of ARP Announcements.
4. Although Zeroconf requires an underlying network that supports ARP (RFC 826), we identified some cases where Zeroconf does not conform to RFC 826.
5. It is not exactly clear in which situations a host may defend its address.

Faithfulness and Traceability We have shown that UPPAAL is able to model Zeroconf faithfully. Basically, for each transition in the model we can point towards a corresponding piece of text in the RFC. The relationships between our model and the RFC have been described in great detail in this chapter, including the design choices and abstractions that we made. Following [BM04], our aim has been to make the model construction *transparent*, so that our model may be more easily understood and checked by others, making its quality measurable in (at least) an informal sense.

We see at least three ways in which UPPAAL can be improved to allow for even more faithful/realistic modeling of Zeroconf and better traceability: (1) An extension with probabilities, along the lines of PRISM [KNP04], is clearly desirable. (2) UPPAAL supports modeling of systems that are described as networks of a *fixed* number of automata with a *fixed* communication structure. This modeling approach does not fit very well with the highly dynamic structure of Zeroconf networks where hosts may join and leave, subnetworks may be joined, etc. (3) To support traceability it would help to add a feature to UPPAAL by which comments are displayed when a user clicks on (or points at) a transition. Items (1) and (2) require a major research effort, whereas item (3) should be easy to implement.

Complexity and Tractability The formal model of Zeroconf that we presented in Section 7.2 cannot be analyzed by UPPAAL for interesting instances with 3 or more hosts. The full version of this chapter is reported in [GVZ06a] and it includes a short manual proof of mutual exclusion for the model that we considered (no message loss, host failure and merging of networks). In order to verify a system with 3 hosts, we had to apply some drastic abstractions. We have argued informally that these abstractions are sound.

A challenging question for us is to come up with (automatically generated) additional abstractions that allow for the automated analysis of larger instances of the protocol. One possibility here would be to try to apply the technique of counterexample guided abstraction refinement [CGJ+00, CFH+03]. A basic idea in the design of Zeroconf is that it does not harm to send additional ARP messages; they have only been added because they may help to ensure (or restore) mutual exclusion in the case of faults. Thus far, we have not been able to come up with abstractions that capture this idea. It is highly desirable to extend UPPAAL with (semi-)automatic support for proving correctness of abstractions. Only abstractions can bridge the gap between realistic and tractable models.

Future Work We have only modelled/analyzed a few simple instances of a part of Zeroconf in a restrictive setting without faulty nodes, merging of subnetworks, etc. So clearly, there are many directions in which our modeling effort can be extended. The timing behavior of Zeroconf becomes really interesting when studied within a setting in which also the probabilistic behavior is modelled. The performance analysis of Zeroconf reported in [BSHV03, KNPS03] has been carried out for an abstract probabilistic model of Zeroconf. A challenging question is whether these results also hold for a (probabilistic extension) of our more realistic model.

Acknowledgments

We thank Peter van der Stok (Philips Research) for suggesting the problem to us, Stuart Cheshire (Apple Computer, Inc.) and Boris Cobbelens (Free University, Amsterdam) for answering all our questions about Zeroconf. Martijn Hendriks, Jasper Berendsen, Jozef Hooman and the students of the Analysis of Embedded Systems course in Nijmegen commented on earlier versions and came with modeling suggestions. Martijn also helped with UPPAAL and noted the occurrence of data saturation. Guy Leduc, Hubert Garavel, Judi Romijn and Ken Turner commented on the use of formal description languages within protocol standards.

Bibliography

[ABV94] L. Aceto, B. Bloom, and F.W. Vaandrager. Turning SOS rules into equations. *Information and Computation*, 111(1):1–52, May 1994.

[ACD90] R. Alur, C. Courcoubetis, and D. Dill. Model Checking for Real-Time Systems. In *IEEE Symposium on Logic in Computer Science*, pages 414–425, 1990.

[ACD93] R. Alur, C. Courcoubetis, and D.L. Dill. Model checking in dense real time. *Information and Computation*, 104:2–34, 1993.

[ACH$^+$95] R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J.Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.

[ACM02] Eugene Asarin, Paul Caspi, and Oded Maler. Timed regular expressions. *J. ACM*, 49(2):172–206, 2002.

[AD94] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[ÅF99] K. Åkesson and M. Fabian. Implementing supervisory control for chemical batch processes. In *International Conf. on Control Applications*, pages 1272–1277. IEEE Computer Society Press, 1999.

[AH94a] R. Alur and T.A. Henzinger. Real-time system = discrete system + clock variables. In T. Rus and C. Rattray, editors, *Theories and Experiences for Real-Time System Development — Papers presented at First AMAST Workshop on Real-Time System Development*, Iowa City, Iowa, November 1993, pages 1–29. World Scientific, 1994.

[AH94b] Rajeev Alur and Thomas A. Henzinger. A really temporal logic. *J. ACM*, 41(1):181–204, 1994.

[AHLP00] R. Alur, T.A. Henzinger, G. Lafferriere, and G.J. Pappas. Discrete abstractions of hybrid systems. *Proc. IEEE*, 88:971–984, 2000.

[AJ95] Luca Aceto and Alan S. A. Jeffrey. A complete axiomatization of timed bisimulation for a class of timed regular behaviours (revised version). *Theoretical Computer Science*, 152(2):251–268, December 1995.

[AL94] Martín Abadi and Leslie Lamport. An old-fashined recipe for real-time. *ACM Trans. Program. Lang. Syst.*, 16(5):1543–1571, 1994.

[Alb02] S. Albert. CYBERNETIX case study informal description, 2002. Available through URL`http://ametist.cs.utwente.nl`.

[AM04] Rajeev Alur and P. Madhusudan. Decision problems for timed automata: A survey. In Bernardo and Corradini [BC04], pages 1–24.

[Asa04] Eugene Asarin. Challenges in timed languages: from applied theory to basic theory. *Bulletin of the EATCS*, 83:106–120, 2004.

[ATM05] Rajeev Alur, Salvatore La Torre, and P. Madhusudan. Perturbed timed automata. In Manfred Morari and Lothar Thiele, editors, *HSCC*, volume 3414 of *Lecture Notes in Computer Science*, pages 70–85. Springer, 2005.

[BC04] Marco Bernardo and Flavio Corradini, editors. *Formal Methods for the Design of Real-Time Systems, International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM-RT 2004, Bertinoro, Italy, September 13-18, 2004, Revised Lectures*, volume 3185 of *Lecture Notes in Computer Science*. Springer, 2004.

[BCG88] M.C. Browne, E.M. Clarke, and O. Grumberg. Characterizing finite Kripke structures in propositional temporal logic. *Theoretical Computer Science*, 59(1,2):115–131, 1988.

[BCH$^+$05] Béatrice Bérard, Franck Cassez, Serge Haddad, Didier Lime, and Olivier H. Roux. Comparison of the expressiveness of timed automata and time petri nets. In Pettersson and Yi [PY05], pages 211–225.

[BCM$^+$92] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking 10^{20} states and beyond. *I&C*, 98(2):142–170, June 1992.

[BCM05] Patricia Bouyer, Fabrice Chevalier, and Nicolas Markey. On the expressiveness of tptl and mtl. In R. Ramanujam and Sandeep Sen, editors, *FSTTCS*, volume 3821 of *Lecture Notes in Computer Science*, pages 432–443. Springer, 2005.

[BDFP00] Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury, and Antoine Petit. Expressiveness of updatable timed automata. In Mogens Nielsen and Branislav Rovan, editors, *MFCS*, volume 1893 of *Lecture Notes in Computer Science*, pages 232–242. Springer, 2000.

[BDHK04] H. Bohnenkamp, P.R. D'Argenio, H. Hermanns, and J.-P. Katoen. MoDeST: A compositional modeling formalism for real-time and stochastic systems. CTIT Tech. Rep. 04-46, University of Twente, 2004. Submitted for publication.

[BDL$^+$01] G. Behrmann, A. David, K.G. Larsen, O. Möller, P. Pettersson, and Wang Yi. UPPAAL – present and future. In *Proceedings of 40*th *IEEE Conference on Decision and Control*. IEEE Press, 2001.

[BDL04] G. Behrmann, A. David, and K.G. Larsen. A tutorial on Uppaal. In Marco Bernardo and Flavio Corradini, editors, *Formal Methods for the Design of Real-Time Systems, International School on Formal Methods for the Design of Computer, Communication and Software Systems (SFM-RT 2004),* Bertinoro, Italy, September 13-18*, Revised Lectures*, volume 3185 of *Lecture Notes in Computer Science*, pages 200–236. Springer, 2004.

[BDM$^+$98] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. KRONOS: A model-checking tool for real-time systems. In A.J. Hu and M. Vardi, editors, *Proceedings of the 10th CAV*, volume 1427 of *LNCS*, pages 546–550. Springer, 1998.

[BFK$^+$98] Howard Bowman, Giorgio P. Faconti, Joost-Pieter Katoen, Diego Latella, and Mieke Massink. Automatic verification of a lip-synchronisation protocol using uppaal. *Formal Asp. Comput.*, 10(5-6):550–575, 1998.

[BG92] G. Berry and G. Gonthier. The ESTEREL synchronous programming language: design, semantics, implementation. *Science of Computer Programming*, 19(2):87–152, November 1992.

[BG06] H. Bowman and R.S. Gomez. *Concurrency Theory, Calculi and Automata for Modelling Untimed and Timed Concurrent Systems.* Springer, January 2006.

[BGK$^+$02] Johan Bengtsson, W. O. David Griffioen, Kåre J. Kristoffersen, Kim Guldstrand Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. Automated verification of an audio-control protocol using uppaal. *J. Log. Algebr. Program.*, 52-53:163–181, 2002.

[BGM02] M. Bozga, S. Graf, and L. Mounier. IF-2.0: A validation environment for component-based real-time systems. In *Proceeding of CAV'02*, volume 2404 of *LNCS*, pages 343–348, Copenhagen, Denmark, 2002. Springer.

[BGO$^+$04] Marius Bozga, Susanne Graf, Ileana Ober, Iulian Ober, and Joseph Sifakis. The if toolset. In Bernardo and Corradini [BC04], pages 237–267.

[BGS00] S. Bornot, G. Gößler, and J. Sifakis. On the construction of live timed systems. In *6th Proceeding of TACAS'00*, volume 1785 of *LNCS*, pages 172–202. Springer-Verlag, 2000.

[BGS05] Howard Bowman, Rodolfo Gómez, and Li Su. A tool for the syntactic detection of zeno-timelocks in timed automata. *Electr. Notes Theor. Comput. Sci.*, 139(1):25–47, 2005.

[BJMY02] Marius Bozga, Hou Jianmin, Oded Maler, and Sergio Yovine. Verification of asynchronous circuits using timed automata. *Electr. Notes Theor. Comput. Sci.*, 65(6), 2002.

[BK82] J.A. Bergstra and J. W. Klop. Fixed point semanitcs in process algebra. Technical Report IW 208, CWI, Amsterdam, 1982.

[BLL$^+$96] J. Bengtsson, K.G. Larsen, F. Larsson, P. Pettersson, and Wang Yi. UPPAAL - A tool suite for automatic verification of real-time systems. In R. Alur, T.A. Henzinger, and E.D. Sontag, editors, *Hybrid Systems III: Verification and Control*, volume 1066 of *LNCS*, pages 232–243. Springer, 1996.

[BM02] J.C.M. Baeten and C.A. Middelburg. *Process Algebra with Timing*. EATCS Monographs. Springer, 2002.

[BM04] E. Brinksma and A. Mader. On verification modelling of embedded systems. Technical Report TR-CTIT-04-03, Centre for Telematics and Information Technology, Univ. of Twente, The Netherlands, January 2004.

[Bou03] Patricia Bouyer. Untameable timed automata! In Helmut Alt and Michel Habib, editors, *STACS*, volume 2607 of *Lecture Notes in Computer Science*, pages 620–631. Springer, 2003.

[Bow99] Howard Bowman. Modelling timeouts without timelocks. In *Proceeding of ARTS'99*, LNCS, page 20. Springer-Verlag, 1999.

[Bow01] Howard Bowman. Time and action lock freedom properties for timed automata. In Myungchul Kim, Byoungmoon Chin, Sungwon Kang, and Danhyung Lee, editors, *FORTE*, volume 197 of *IFIP Conference Proceedings*, pages 119–134. Kluwer, 2001.

[Bra96] B. A. Brandin. The real-time supervisory control of an experimental manufacturing cell. In *IEEE Transactions on Robotics and Automation*, volume 12, pages 1–13, 1996.

[BS98a] S. Bornot and J. Sifakis. On the composition of hybrid systems. In Thomas A. Henzinger and Shankar Sastry, editors, *Hybrid Systems: Computation and Control, First International Workshop, HSCC'98*, volume 1386 of *LNCS*, pages 49–63. Springer, 1998.

[BS98b] G. Bruns and M.G. Staskauskas. Applying formal methods to a protocol standard and its implementations. In *Proceedings International Symposium on Software Engineering for Parallel and Distributed Systems (PDSE 1998)*, 20-21 April, 1998, Kyoto, Japan, pages 198–205. IEEE Computer Society, 1998.

[BS00] S. Bornot and J. Sifakis. An algebraic framework for urgency. *Information and Computation*, 163:172–202, 2000.

[BSHV03] H. Bohnenkamp, P. van der Stok, H. Hermanss, and F.W. Vaandrager. Cost-optimisation of the IPv4 zeroconf protocol. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN2003)*, pages 531–540, Los Alamitos, California, 2003. IEEE Computer Society.

[BST98] S. Bornot, J. Sifakis, and S. Tripakis. Modeling urgency in timed systems. In Roever W.-P. de, H. Langmaack, and A. Pnueli, editors, *Compositionality: The Significant Difference*, volume 1536 of *LNCS*, pages 103–129. Springer, 1998.

[BT04] Roberto Barbuti and Luca Tesei. Timed automata with urgent transitions. *Acta Inf.*, 40(5):317–347, 2004.

[BY03] Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In Jörg Desel, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Lectures on Concurrency and Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 87–124. Springer, 2003.

[CAG05] S. Cheshire, B. Aboba, and E. Guttman. Dynamic configuration of ipv4 link-local addresses (rfc 3927), May 2005. `http://www.ietf.org/rfc/rfc3927.txt`.

[CE81] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In Dexter Kozen, editor, *Logic of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1981.

[Čerāns92] K. Čerāns. Decidability of bisimulation equivalences for parallel timer processes. In G. von Bochmann and D.K. Probst, editors, *Proceedings of the 4th CAV*, volume 663 of *LNCS*, pages 302–315. Springer, 1992.

[CFH$^+$03] Edmund M. Clarke, Ansgar Fehnker, Zhi Han, Bruce H. Krogh, Joël Ouaknine, Olaf Stursberg, and Michael Theobald. Abstraction and counterexample-guided refinement in model checking of hybrid systems. *Int. J. Found. Comput. Sci.*, 14(4):583–604, 2003.

[CGH$^+$93] E. M. Clarke, O. Grumberg, H. Hiraishi, S. Jha, D. E. Long, K. L. McMillan, and L. A. Ness. Verification of the Futurebus+ cache coherence protocol. In *Proc. CHDL*, pages 15–30, 1993.

[CGJ$^+$00] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In E. Allen Emerson and A. Prasad Sistla, editors, *CAV*, volume 1855 of *Lecture Notes in Computer Science*, pages 154–169. Springer, 2000.

[Che06] S. Cheshire. Personal communication, February 2006.

[CL99] C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. KLuwer Academic, 1999.

[CL00] Franck Cassez and Kim Guldstrand Larsen. The impressive power of stopwatches. In Catuscia Palamidessi, editor, *CONCUR*, volume 1877 of *Lecture Notes in Computer Science*, pages 138–152. Springer, 2000.

[Cor98] F. Corradini. On performance congruences for process algebras. *Information and Computation*, 145(2):191–230, 1998.

[CS05] S. Cheshire and D.H. Steinberg. *Zero Configuration Networking: The Definite Guide*. O'Reilly Media, Inc., 2005.

[D'A99] P.R. D'Argenio. *Algebras and Automata for Timed and Stochastic Systems*. PhD thesis, Department of Computer Science, University of Twente, November 1999.

[DB96] P.R. D'Argenio and E. Brinksma. A calculus for timed automata (Extended abstract). In B. Jonsson and J. Parrow, editors, *Procs. of FTRTFT'96,* Uppsala, Sweden, volume 1135 of *LNCS*, pages 110–129. Springer, 1996.

[DG05] P.R. D'Argenio and B. Gebremichael. The coarsest congruence for timed automata with deadlines contained in bisimulation. In *16th International Conference on Concurrency Theory (CONCUR05)*, volume 3653 of *LNCS*, pages 125–140, San francisco, USA, August 2005.

[DG06] P.R. D'Argenio and B. Gebremichael. Axiomatizing timed automata with deadlines. Technical Report ICIS-R06xxx, ICIS - Radboud University Nijmegen, 2006.

[DHKK01] P.R. D'Argenio, H. Hermanns, J.-P. Katoen, and R. Klaren. MoDeST - a modelling and description language for stochastic timed systems. In L. de Alfaro and S. Gilmore, editors, *Proceedings of PAPM-PROBMIV 2001*, volume 2165 of *LNCS*, pages 87–104. Springer, 2001.

[Dij74] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications ACM*, 17:643–644, 1974.

[Dil89] David L. Dill. Timing assumptions and verification of finite-state concurrent systems. In Joseph Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*, pages 197–212. Springer, 1989.

[Dim01] C. Dima. *An algebric theory of real-time formal languages*. PhD thesis, Université Joseph Fourier, Grenoble, France, 2001.

[Feh99] Ansgar Fehnker. Scheduling a Steel Plant with Timed Automata. In *Sixth International Conference on Real-Time Computing Systems and Applications (RTCSA' 99).* IEEE Computer Society Press, 1999.

[FZ95] C. J. Fidge and J. J. Zic. An expressive real-time CCS. In *Second Australasian Conference on Parallel and Real-Time Systems (PART'95)*. Fremantle, September 1995.

[GB03] R. Gomez and H. Bowman. Discrete Timed Automata and MONA: Description, Specification and Verification of a Multimedia Stream. In *Proceeding of FORTE'03*, volume 2767 of *LNCS*, pages 177–192. Springer, 2003.

[GHKU03] B. Gebremichael, H. Hermanns, T. Krilavičius, and Y.S. Usenko. Hybrid modeling of a vehicle surveillance system with real-time data processing. In *Proc. Int. Conf. on Dynamical Systems Modeling and Stability Investigation*, page 419, Kyiv, Ukraine, May 2003.

[GKU04] B. Gebremichael, T. Krilavičius, and Y.S. Usenko. A formal analysis of a car periphery supervision system. In J. Zaytoon, V. Carre-Mennetrier, and X. Cao, editors, *Seventh International Workshop on Discrete Event Systems WODES'04*, pages 433–439, Reims, France, September 2004. Elsevier Science Ltd. Also available as Technical Report NIII-R0418, NIII, University of Nijmegen.

[Gla93] R.J. van Glabbeek. A complete axiomatization for branching bisimulation congruence of finite-state behaviours. In *Proc. MFCS'93*, volume 711 of *LNCS*, pages 473–484. Springer, 1993.

[GLTV03] S.J. Garland, N.A. Lynch, J. Tauber, and M. Vaziri. IOA user guide and reference manual, 2003.

[GO01] S. Graf and I. Ober. A real-time profile for UML and how to adapt it to SDL. In *Proceeding of SDL Forum'03*, volume 2165 of *LNCS*, pages 55–76. Springer, 2001.

[GS05a] Gregor Gössler and Joseph Sifakis. Composition for component-based modeling. *Sci. Comput. Program.*, 55(1-3):161–183, 2005.

[GS05b] Gregor Gössler and Joseph Sifakis. Composition for component-based modeling. *Sci. Comput. Program.*, 55(1-3):161–183, 2005.

[Gun97] J. Gunnarsson. *Symbolic Methods and Tools for Discrete Event Dynamic Systems*. PhD thesis, Linköping Studies in Science and Technology, 1997.

[GV03a] B. Gebremichael and F.W. Vaandrager. Control synthesis for a smart card personalization system using symbolic model checking. In *Proceedings First International Workshop on Formal Modeling and Analysis of Timed Systems (FORMATS'03)*, volume 2791 of *LNCS*, Marseille, France, September 2003. Springer Verlag.

[GV03b] Biniam Gebremichael and Frits Vaandrager. Control synthesis for a smart card personalization system using symbolic model checking. Technical Report NIII-R0312, Computing Science Institute, University of Nijmegen, 2003.

[GV04] B. Gebremichael and F. Vaandrager. Specifying urgency in timed I/O automata. Technical Report NIII-R0459, ICIS. Radboud University Nijmegen, 2004.

[GV05] B. Gebremichael and F.W. Vaandrager. Specifying urgency in timed I/O automata. In *Proceeding of 3rd IEEE International Conference on Software Engineering and Formal Methods (SEFM05)*, pages 64–73, Koblenz, Germany, September 2005. IEEE Computer Society.

[GVZ06a] B. Gebremichael, F.W. Vaandrager, and M. Zhang. Analysis of a protocol for dynamic configuration of IPv4 link local addresses using UPPAAL. Technical Report ICIS-R06016, ICIS, Radboud University Nijmegen, 2006.

[GVZ06b] B. Gebremichael, F.W. Vaandrager, and M. Zhang. Analysis of the zeroconf protocol using UPPAAL. In *Proceedings 6th Annual ACM Conference on Embedded Software (EMSOFT)*. ACM Press, October 2006. To appear.

[HBL$^+$03] Martijn Hendriks, Gerd Behrmann, Kim Guldstrand Larsen, Peter Niebert, and Frits W. Vaandrager. Adding symmetry reduction to uppaal. In Kim Guldstrand Larsen and Peter Niebert, editors, *FORMATS*, volume 2791 of *Lecture Notes in Computer Science*, pages 46–59. Springer, 2003.

[Hen96] T.A. Henzinger. The theory of hybrid automata. In *Proc. 11th LICS'96*, pages 278–292, New Brunswick, New Jersey, USA, 1996.

[Hen98] Thomas A. Henzinger. It's about time: Real-time logics reviewed. In Davide Sangiorgi and Robert de Simone, editors, *CONCUR*, volume 1466 of *Lecture Notes in Computer Science*, pages 439–454. Springer, 1998.

[Her02] H. Hermanns. *Interactive Markov Chains : The Quest for Quantified Quality*, volume 2428 of *LNCS*. Springer, 2002.

[HH95] T.A. Henzinger and P.-H. Ho. Algorithmic analysis of nonlinear hybrid systems. In P. Wolper, editor, *Proc. 7th CAV'95*, volume 939, pages 225–238. Springer, 1995.

[HHWT97] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. HYTECH: A model checker for hybrid systems. *Int. Journal on Software Tools for Technology Transfer*, 1:110–122, 1997.

[Hil96] J. Hillston. *A Compositional Approach to Performance Modelling*. Distinguished Dissertation in Computer Science. Cambridge University Press, 1996.

[HKWT95] Thomas A. Henzinger, Peter W. Kopke, and Howard Wong-Toi. The expressive power of clocks. In Zoltán Fülöp and Ferenc Gécseg, editors, *ICALP*, volume 944 of *Lecture Notes in Computer Science*, pages 417–428. Springer, 1995.

[HL02] M. Hendriks and K.G. Larsen. Exact acceleration of real-time model checking. *ENTCS*, 65(6): , April 2002.

[HLP01] Thomas Hune, Kim G. Larsen, and Paul Pettersson. Guided Synthesis of Control Programs usingUPPAAL. *Nordic Journal of Computing*, 8(1):43–64, 2001.

[HMP92] Thomas A. Henzinger, Zohar Manna, and Amir Pnueli. What good are digital clocks? In Werner Kuich, editor, *ICALP*, volume 623 of *Lecture Notes in Computer Science*, pages 545–558. Springer, 1992.

[HNSY94] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.

[Hoa78] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21(8):666–677, 1978.

[Hol03] G.J. Holzmann. *The Spin model checker: primer and reference manual*. Addison-Wesley, 2003.

[HPR94] N. Halbwachs, Y. Proy, and P. Raymond. Verification of linear hybrid systems by means of convex approximations. In *Proc. 1st. SAS'94*, LNCS 864, pages 223–237, Namur, Belgium, September 1994. Springer.

[HVG03] Jinfeng Huang, Jeroen Voeten, and Marc Geilen. Real-time property preservation in approximations of timed systems. In *MEMOCODE*, pages 163–171. IEEE Computer Society, 2003.

[ID93] C. Norris Ip and David L. Dill. Better verification through symmetry. In David Agnew, Luc J. M. Claesen, and Raul Camposano, editors, *CHDL*, volume A-32 of *IFIP Transactions*, pages 97–111. North-Holland, 1993.

[IKL$^+$00] T.K. Iversen, K.J. Kristoffersen, K.G. Larsen, M. Laursen, R.G. Madsen, S.K. Mortensen, P. Pettersson, and C.B. Thomasen. Model-checking real-time control programs — Verifying LEGO mindstorms systems using UPPAAL. In *IEEE Euromicro Conf. on Real-Time Systems*, pages 147–155, 2000.

[J.L96] J.L.Lions. Ariane 5 flight 501 failure: Report of the inquiry board, July 1996. Available at `http://www.esa.int`.

[JLS00] H.E. Jensen, K.G. Larsen, and A. Skou. Scaling up UPPAAL automatic verification of real-time systems using compositionality and abstraction. In Mathai Joseph, editor, *FTRTFT 2000 proceedings*, number 1926 in LNCS, pages 19–30. Springer-Verlag, 2000.

[Jon97] Mike Jones. What really happened on mars rover pathfinder, 1997. The Risks Digests 19:49.

[KLM04] D.K. Kaynar, N.A. Lynch, and S. Mitra. Specifying and proving timing properties with TIOA tools. In *25th Proceeding of RTSS'04 WIP*. IEEE Computer Society, 2004.

[KLSV03a] D.K. Kaynar, N.A. Lynch, R. Segala, and F.W. Vaandrager. A framework for modelling timed systems with restricted hybrid automata. In *24th Proceedings of RTSS'03*, pages 166–178. IEEE Computer Society Press, 2003.

[KLSV03b] D.K. Kaynar, N.A. Lynch, R. Segala, and F.W. Vaandrager. The theory of timed I/O automata. Technical Report MIT-LCS-TR-917, MIT Laboratory for Computer Science, Cambridge, MA, 2003.

[KNP04] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM 2.0: A tool for probabilistic model checking. In *Proceedings of the 1st International Conference on Quantitative Evaluation of Systems (QEST04)*, pages 322–323. IEEE Computer Society, 2004.

[KNPS03] M. Kwiatkowska, G. Norman, D. Parker, and J. Sproston. Performance analysis of probabilistic timed automata using digital clocks. In K. Larsen and P. Niebert, editors, *Proc. Formal Modeling and Analysis of Timed Systems (FORMATS'03)*, volume 2791 of *LNCS*, pages 105–120. Springer-Verlag, 2003.

[Koy90] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

[KPSY99] Yonit Kesten, Amir Pnueli, Joseph Sifakis, and Sergio Yovine. Decidable integration graphs. *Inf. Comput.*, 150(2):209–243, 1999.

[KR03] S. Kowalewski and M. Rittel. IST Project AMETIST: Real-time allocation for car periphery supervision. Technical report, Robert Bosch GmbH, 2003. Preliminary Description (Deliverable No. 3.1.3) see `http://ametist.cs.utwente.nl/`.

[KU03] Tomas Krilavicius and Yaroslav Usenko. Smart card personalisation machine inUPPAAL and μCRL, 2003. In preparation.

[Lam83] L. Lamport. What good is temporal logic? In R.E. Mason, editor, *Information Processing 83*, pages 657–668. North-Holland, 1983.

[Lam05] Leslie Lamport. Real-time model checking is really simple. In Dominique Borrione and Wolfgang J. Paul, editors, *CHARME*, volume 3725 of *Lecture Notes in Computer Science*, pages 162–175. Springer, 2005.

[LMN05] Kim G. Larsen, Marius Mikucionis, and Brian Nielsen. Testing real-time embedded software using UPPAAL-TRON: an industrial case study. In *the 5th ACM International Conference on Embedded Software*, pages 299 – 306. ACM Press New York, NY, USA, September 18–22 2005.

[Loe02] V. Loeb. 'Friendly Fire' deaths traced to dead battery: Taliban targeted, but US forces killed. *Washington Post*, page 21, March 2002.

[LPY97] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1(1–2):134–152, October 1997.

[LSV03] N.A. Lynch, R. Segala, and F.W. Vaandrager. Hybrid I/O automata. *Information and Computation*, 185(1):105–157, 2003.

[LT87] Nancy A. Lynch and Mark R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing*, pages 137–151. MIT, 1987.

[LW93] Y. Li and W.M. Wonham. Control of vector discrete-event systemsI - the base model. In *IEEE Trans. on Automatic Control*, volume 38, pages 1214–1227. IEEE Computer Society Press, 1993.

[LY02] Huimin Lin and Wang Yi. Axiomatizing timed automata. *Acta Informatica*, 38(4):277–305, 2002.

[Mad04] A. Mader. Deriving schedules for a smart card personalisation system, 2004.

[McM93] K.L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem*. Kluwer Academic Publishers, 1993.

[Mer74] P. M. Merlin. *A study of the recoverability of computing systems*. PhD thesis, University of California, irvine, CA, 1974.

[Mil84] R. Milner. A complete inference system for a class of regular behaviours. *J. of Comp. and System Sci.*, 28:439–466, 1984.

[Mil89a] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[Mil89b] R. Milner. A complete axiomatisation for observational congruence of finite-state behaviours. *Information and Computation*, 81(2):227–247, 1989.

[MMT91] M. Merritt, F. Modugno, and M. Tuttle. Time constrained automata. In *6th Proceeding of CONCUR'91*, volume 527 of *LNCS*, pages 408–423. Springer-Verlag, 1991.

[MNP05] Oded Maler, Dejan Nickovic, and Amir Pnueli. Real time temporal logic: Past, present, future. In Pettersson and Yi [PY05], pages 2–16.

[Mor00] R. Moritz. Pre-crash sensing - functional. Technical report, Robert Bosch GmbH, 2000.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer, 1992.

[MP95] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer, 1995.

[NS91] Xavier Nicollin and Joseph Sifakis. An overview and synthesis on timed process algebras. In Kim Guldstrand Larsen and Arne Skou, editors, *CAV*, volume 575 of *Lecture Notes in Computer Science*, pages 376–398. Springer, 1991.

[NY00] Peter Niebert and Sergio Yovine. Computing optimal operation schemes for multi batch operation of chemical plants. In

Nancy A. Lynch and Bruce H. Krogh, editors, *Hybrid Systems: Computation and Control, Third International Workshop,HSCC'00*, volume 1790 of *Lecture Notes in Computer Science*, pages 338–351. Springer, 2000.

[Plu82] David C. Plummer. An ethernet address resolution protocol (rfc 826), November 1982. `http://www.ietf.org/rfc/rfc826.txt`.

[PY05] Paul Pettersson and Wang Yi, editors. *Formal Modeling and Analysis of Timed Systems, Third International Conference, FORMATS 2005, Uppsala, Sweden, September 26-28, 2005, Proceedings*, volume 3829 of *Lecture Notes in Computer Science*. Springer, 2005.

[Ram74] C. Ramchandani. *Analysis of asynchronous concurrent systems by timed petri nets*. PhD thesis, Massachusetts Institute of Technology, Camvridge, MA, 1974.

[Ras99] J.-F. Raskin. *Logics, Automata and Classical Theories for Deciding Real-Time*. PhD thesis, Univ. Namur, Namur, Belgium, 1999.

[Rob04] C.M. Robson. TIOA and UPPAAL. Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, 2004.

[Rom04] J.M.T. Romijn. Improving the quality of protocol standards: Correcting IEEE 1394.1 FireWire net update. *Nieuwsbrief van de Nederlandse Vereniging voor Theoretische Informatica*, 8:23–30, 2004. Available at `http://www.win.tue.nl/oas/index.html?iqps/`.

[Ruy03] T. C. Ruys. Optimal scheduling using branch and bound with SPIN 4.0. In *Proceedings of SPIN 2003, the 10th SPIN workshop*, 2003. To appear.

[RW89] P.J.G. Ramadge and W.M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77:81–98, 1989.

[SGSAL98] R. Segala, R. Gawlick, J.F. Søgaard-Andersen, and N.A. Lynch. Liveness in timed and untimed systems. *Information and Computation*, 141(2):119–171, 1998.

[SK91] R.S. Sreenivas and B.H. Krogh. On condition/event systems with discrete state realizations. In *Discrete Event Dynamic Systems. Theory and Applications 1*, pages 209–236. Flumer Academic, 1991.

[Smi85] Brian Cantwell Smith. The limits of correctness. *ACM SIGCAS Computers and Society*, 14:18–26, 1985.

[SSL$^+$95] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of discrete event systems. *IEEE Transactions on Automatic Control*, 40(9):1555– 1575, September 1995.

[Sto03] Mariëlle Stoelinga. Fun with firewire: A comparative study of formal verification methods applied to the IEEE 1394 root contention protocol. *Formal Asp. Comput.*, 14(3):328–337, 2003.

[SV96] Jan Springintveld and Frits W. Vaandrager. Minimizable timed automata. In Bengt Jonsson and Joachim Parrow, editors, *FTRTFT*, volume 1135 of *Lecture Notes in Computer Science*, pages 130–147. Springer, 1996.

[SY96] J. Sifakis and S. Yovine. Compositional specification of timed systems. In *Proceedings of the 13th Annual Symp. on Theoretical Aspects of Computer Science, STACS'96*, volume 1046 of *LNCS*, pages 347–359, Grenoble, France, 1996. Springer.

[Tel94] G. Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, 1994.

[TFF$^+$01] S. Thiel, S. Ferber, T. Fischer, A. Hein, and M. Schlick. A case study in applying a product line approach for car periphery supervision system. Technical report, Robert Bosch GmbH, 2001.

[TY01] S. Tripakis and S. Yovine. Analysis of timed systems using time-abstracting bisimulations. *Formal Methods in System Design*, 18(1):25–68, 2001.

[vLRG03] Izak van Langevelde, Judi Romijn, and Nicolae Goga. Founding firewire bridges through promela prototyping. In 8th *International Workshop on Formal Methods for Parallel Programming: Theory and Applications (FMPPTA)*. IEEE Computer Society Press, April 2003.

[Wan04] F. Wang. Formal verification of timed systems: A survey and perspective. *Proceedings of the IEEE*, 92(8):1283–1305, 2004.

[Wei03] G. Weiss. LSCs modeling and cybernetixCS, 2003. to appear.

[Yi91] Wang Yi. Ccs + time = an interleaving model for real time systems. In Javier Leach Albert, Burkhard Monien, and Mario Rodríguez-Artalejo, editors, *ICALP*, volume 510 of *Lecture Notes in Computer Science*, pages 217–228. Springer, 1991.

[YL94] W.M. Wonham Y. Li. Control of vector discrete-event systemsII - controller synthesis. In *IEEE Transactions on Autom. Control*, volume 39, pages 512–531, 1994.

[Yor00] K. Yorav. *Exploiting Syntactic Structure for Automatic Verification*. PhD thesis, TECHNION - Israel Institute of Technology, June 2000.

[Yov98] S. Yovine. Model checking timed automata. In G. Rozenberg and F.W. Vaandrager, editors, *Lectures on Embedded Systems*, volume 1494 of *LNCS*, pages 114–152. Springer, 1998.

[ZW01] Z. Zhang and W.M. Wonham. STCT: An efficient algorithm for supervisory control design. In *Symposium on Supervisory Control of Discrete Event Systems*, 2001.

Samenvatting (Dutch summary)

Real-time systemen zijn computersystemen waarvan de correctheid niet alleen afhangt van de (logische) uitkomst van een berekening, maar ook van het moment waarop deze uitkomst beschikbaar komt. Typische voorbeelden van real-time systemen zijn besturingen van spoorwegovergangen, systemen ter ondersteuning van luchtverkeersleiding, en multimedia netwerken. Real-time systemen zijn meestal ingebed in grotere systemen die weer in interactie staan met hun omgeving. Zo vormt de computerbesturing van een spoorwegovergang een onderdeel van een groter veiligheidsysteem dat verantwoordelijk is voor het sluiten en openen van de overgang wanneer er een trein langskomt.

Door concurrentie tussen bedrijven en de noodzaak functionaliteit en prestaties te verbeteren met zo min mogelijk hardware, neemt de complexiteit van real-time toepassingen zeer snel toe. Dit leidt tot een toenemend aantal fouten in deze systemen, waardoor de noodzaak om correctheid nauwkeurig te specificeren en rigoreus te verifiëren groter wordt dan ooit. Formele specificatie en verificatie is buitengewoon succesvol gebleken in de VLSI industrie. Er zijn veelbelovende resultaten die suggereren dat een vergelijkbaar succes ook mogelijk is voor real-time systemen.

Een van deze resultaten is de introductie van getimede automaten voor het formeel specificeren en verifiëren van de correctheid van real-time systemen. Getimede automaten worden ondersteund door een sterke theoretische fundering en efficiënt verificatiegereedschap zoals de model checker UPPAAL. Succesvolle toepassing van formele verificatie vereist een taal waarin enerzijds de te analyseren systemen accuraat gemodelleerd kunnen worden terwijl anderzijds verificatie toch mogelijk is. Dit proefschrift onderzoekt de expressiviteit van getimede automaten zowel in theorie als in de praktijk. Het theoretische onderzoek heeft betrekking op noties van equivalentie tussen getimede automaten en vertalingen tussen formalismes. Het praktische onderzoek richt zich op de vraag hoe geschikt getimede automaten zijn voor het modelleren en analyseren van (relevante aspecten van) complexe real-time systemen, zoals die ontwikkeld zijn of worden door de industrie.

Dit promotieonderzoek is uitgevoerd in de context en onder de auspiciën van het Europese onderzoeksproject Advanced Methods for Timed

Systems (AMETIST)[2] dat liep van April 2002 tot en met Juni 2005. Het AMETIST consortium bestond uit zeven academische en vier industriële partners uit vijf verschillende landen. Een van de belangrijkste doelstellingen van het project, en tevens het achterliggende doel van dit proefschrift, was *het ontwikkelen van een krachtige modelleermethodologie, gebaseerd op het getimede automaten model, waarmee realistisch, complexe industriële systemen gemodelleerd kunnen worden.*

Overzicht

Dit proefschrift bestaat uit een zestal (technische) hoofdstukken, die gegroepeerd kunnen worden in twee delen. Het eerste deel (Hoofdstuk 2-4) vergelijkt de theoretische expressiviteit van diverse talen voor getimede automaten. Hierbij wordt met name gekeken naar het aspect "tijdsvoortgang": is het mogelijk om in een model de tijd stil te laten staan? Omdat we er van uitgaan dat in de werkelijkheid de tijd altijd voortschrijdt, willen we deze mogelijkheid liefst uitsluiten, zonder de expressiviteit verder in te perken. Drie populaire getimede automatenformalismen worden bestudeerd: getimede automaten met invarianten (Safety Timed Automata), getimede automaten met deadlines (TAD), en getimede input/output automaten (TIOA). In het tweede deel van dit proefschrift (Hoofdstukken 5-7) wordt de "praktische" expressiviteit van getimede automaten bestudeerd aan de hand van complexe, industriële case studies die zijn voorgesteld binnen het AMETIST project. Onze resultaten laten zien dat getimede automatenformalismen in staat zijn om kritieke onderdelen van complexe industriële systemen op een nauwkeurige en natuurlijke wijze te modelleren.

Hieronder vatten we de inhoud van de afzonderlijke hoofdstukken kort samen.

- Hoofdstuk 2 presenteert een nieuwe bisimulatierelatie, *drop bisimulation* genaamd, en lost daarmee een probleem op met het zgn. synchronisation delay dat zich voordoet bij de bestaande bisimulatierelatie voor TADs. De nieuwe bisimulatie wordt gekarakteriseerd in termen van een symbolische bisimulatie, waaruit volgt dat drop bisimulation beslisbaar is.
- Hoofdstuk 3 is een vervolg op Hoofdstuk 2 en presenteert een algebraïsch bewijssysteem dat ons in staat stelt om drop bisimulation equivalentie te bewijzen via syntactische manipulatie. In dit hoofdstuk laten we zien dat het bewijssysteem gezond en volledig is.
- Hoofstuk 4 introduceert een nieuwe manier om deadlines te specificeren in het TIOA formalisme, waardoor specificaties korter en natuurlijker worden. De nieuwe specificatiestijl wordt vergeleken met

[2]`http://ametist.cs.utwente.nl`

andere manieren om deadlines te modelleren en tevens worden vertalingen tussen de specificatieformalismen gedefinieerd.

- Hoofdstuk 5 presenteert een formeel model en analyse van het Car Periphery Supervision (CPS) syteem gebruikmakend van getimede automaten. Het CPS systeem, ontwikkeld door AMETIST partner Robert BOSCH GmbH, is een radarsysteem dat is ingebouwd in een auto waarmee de directe omgeving in de gaten kan worden gehouden. Het CPS systeem is hybride van aard, maar wij zijn er in geslaagd om een passende abstractie te vinden die ons in staat stelde om het systeem te modelleren met getimede automaten en relevante eigenschappen te verifiëren met behulp van de model checker UPPAAL.
- Hoofdstuk 6 rapporteert over een tweede case study, die is voorgesteld door Cybernetix, een andere industriële partner uit het AMETIST project. Deze case study gaat over een machine voor het personaliseren van smart cards. Een van de onderzoeksproblemen waar we aan hebben gewerkt is het afleiden van een besturing die de machine weer in een reguliere toestand kan brengen na het optreden van een fout. Bestaande programma's voor het afleiden van besturingen zijn niet krachtig genoeg voor zulke grote systemen ($> 10^{13}$ toestanden). Daarom hebben we het systeem gemodelleerd gebruikmakend van de model checker SMV en een "truc" bedacht om met behulp van SMV een besturing te genereren waarmee het systeem fouten kan afhandelen. Voor dit probleem was het gebruik van een klassieke (discrete tijd) model checker het meest voor de hand liggend / natuurlijk. Wij denken dat het een interessante uitdaging vormt om discrete tijd en real-time model checkers met elkaar te integreren.
- Hoofdstuk 7 beschrijft een model en analyse van het Zeroconf protocol. Zeroconf maakt het mogelijk om een IPv4 netwerk te configureren zonder gebruik te maken van een externe server zoals DHCP. De bijdrage van dit hoofdstuk is tweeledig. Allereerst beschrijft het een formele specificatie van kritieke onderdelen van het protocol. Ons onderzoek bracht diverse fouten (of op zijn minst ambiguïteiten) aan het licht in de protocolstandaard die nog door niemand anders waren opgemerkt. Ten tweede, zijn wij er in geslaagd relevante eigenschappen te verifiëren uit de specificatie, hetgeen inzicht oplevert ten aanzien van welke eigenschappen gelden onder welke aannames.

Conclusie

Het toestandsexplosieprobleem was en is de belangrijkste uitdaging voor het schaalbaar maken van formele verificatie. Dit probleem doet zich voor ten gevolge van het enorme aantal toestanden van realistische systemen. De laatste jaren is er belangrijke vooruitgang geboekt bij het aanpakken van het toestandsexplosieprobleem. Deze vooruitgang is niet alleen te danken aan de verbetering van computerhardware, waarvan de prestaties iedere 18 maanden verdubbelen (Moore's wet), maar ook aan nieuwe algoritmische technieken zoals abstractie, symbolische analyse en compacte datastructuren. In Hoofdstukken 5, 6 en 7 van dit proefschrift laten we de toenemende kracht van model checkers zien. We maken gebruik van state-of-the-art modelleer- en verificatietechnieken om de gewenste eigenschappen te bewijzen. Zonder gebruik van recent ontwikkelde algoritmische technieken was het ons niet gelukt om dit voor elkaar te krijgen.

Er zijn echter nog steeds veel gevallen waarin het niet mogelijk is om toestandsruimtes geheel te doorzoeken. Een alternatieve en veelbelovende aanpak van dit probleem is compositionele analyse. Efficiënte implementatie van compositionele verificatietechnieken vereist slimme algoritmische technieken. Een sleutel tot succes is echter de expressieve kracht van de modelleertalen, het vermogen om componenten afzonderlijk te specificeren met duidelijk gedefineerde interfaces, en het vermogen om eigenschappen van het systeem af te leiden van eigenschappen van de componenten.

Componentgebaseerde analyse wordt veel gebruikt in de VLSI industrie, maar voor real-time systemen wordt het nog niet echt toegepast. Dit heeft te maken met de moeilijkheid om compositioneel te redeneren over systemen in de aanwezigheid van een globale notie van tijd.

Een kleine maar belangrijke stap richting het compositioneel redeneren over real-time systemen is de aanpak van Sifakis en collega's, waarbij afwezigheid van tijd deadlocks en Zeno gedrag (oneindig veel gebeurtenissen in eindige tijd) gegarandeerd wordt per constructie. De bijdrage van dit proefschrift is het creëren van de mogelijkheid om equivalente componenten voor elkaar te substitueren zonder daarbij het gedrag van het systeem te veranderen (zie Hoofdstuk 2). Dit is een belangrijk hulpmiddel voor componentgebaseerde analyse van real-time systemen. Het probleem of twee componenten equivalent zijn is beslisbaar en een algebraïsche theorie voor het syntactisch afleiden van equivalentie wordt gepresenteerd (Hoofdstuk 3).

Er is nog een lange weg te gaan voordat compositionaliteit volledig benut kan worden bij de analyse van real-time systemen en dit vormt een interessante richting voor toekomstig onderzoek. Zolang het nog niet mogelijk is om bepaalde systemen automatisch te verifiëren, kan het formeel specificeren van deze systemen toch al een belangrijke bijdrage leveren aan het verbeteren van de kwaliteit van het systeem, ook als startpunt voor simulatie en documentatie. Formele specificatie kan inconsistenties en ambiguïteiten detecteren in een vroeg stadium van het ontwerpproces. Het dwingt ontwerpers onduidelijkheden te verhelderen die anders wellicht over het hoofd zouden zijn gezien. Dit geldt in het bijzonder voor de talrijke communicatieprotocollen die zijn gestandaardiseerd door de IEEE SA en andere standaardiseringsinstanties in termen van natuurlijke taal. Deze standaarden bevatten de ambiguïteit die inherent is aan het gebruik van na-

tuurlijke taal. Inmiddels is een aantal artikelen gepubliceerd waarin wordt aangetoond dat formele specificatie kan helpen om de kwaliteit van complexe ontwerpen te verbeteren. Dit proefschrift ondersteunt de these dat formele specificatie hier inderdaad een bijdrage kan leveren. Sommige softwarepaketten voor formele specificatie en verificatie (in het bijzonder UPPAAL) zijn inmiddels krachtig genoeg dat ze hierbij kunnen helpen. Als uitvloeisel van ons onderzoek stellen we in dit proefschrift een aantal verbeteringen en uitbreidingen voor waarmee de "praktische" expressiviteit van specificatietalen (in het bijzonder die van UPPAAL) verder vergroot kan worden.

(Special thanks to Frits Vaandrager, who translated this summary into the Dutch language.)

Curriculum Vitae

1987 – 1990	High School. Asmara, Eritrea.
1991 – 1996	B.Sc. in Mathematics. Asmara Univerity, Eritrea.
1996 – 1997	Graduate Assistant. Asmara Univerity, Eritrea.
1997 – 1999	M.Sc. Computer Science. Uppsala Univerity, Sweden.
2000 – 2002	Assistant Lecturer. Asmara Univerity, Eritrea.
2002 – 2006	Junior researcher. Radboud University Nijmegen, The Netherlands.

Titles in the IPA Dissertation Series

H. ter Doest. *Towards Probabilistic Unification-based Parsing*. Faculty of Computer Science, UT. 1999-02

J.P.L. Segers. *Algorithms for the Simulation of Surface Processes*. Faculty of Mathematics and Computing Science, TUE. 1999-03

C.H.M. van Kemenade. *Recombinative Evolutionary Search*. Faculty of Mathematics and Natural Sciences, UL. 1999-04

E.I. Barakova. *Learning Reliability: a Study on Indecisiveness in Sample Selection*. Faculty of Mathematics and Natural Sciences, RUG. 1999-05

M.P. Bodlaender. *Scheduler Optimization in Real-Time Distributed Databases*. Faculty of Mathematics and Computing Science, TUE. 1999-06

M.A. Reniers. *Message Sequence Chart: Syntax and Semantics*. Faculty of Mathematics and Computing Science, TUE. 1999-07

J.P. Warners. *Nonlinear approaches to satisfiability problems*. Faculty of Mathematics and Computing Science, TUE. 1999-08

J.M.T. Romijn. *Analysing Industrial Protocols with Formal Methods*. Faculty of Computer Science, UT. 1999-09

P.R. D'Argenio. *Algebras and Automata for Timed and Stochastic Systems*. Faculty of Computer Science, UT. 1999-10

G. Fábián. *A Language and Simulator for Hybrid Systems*. Faculty of Mechanical Engineering, TUE. 1999-11

J. Zwanenburg. *Object-Oriented Concepts and Proof Rules*. Faculty of Mathematics and Computing Science, TUE. 1999-12

R.S. Venema. *Aspects of an Integrated Neural Prediction System*. Faculty of Mathematics and Natural Sciences, RUG. 1999-13

J. Saraiva. *A Purely Functional Implementation of Attribute Grammars*. Faculty of Mathematics and Computer Science, UU. 1999-14

R. Schiefer. *Viper, A Visualisation Tool for Parallel Program Construction*. Faculty of Mathematics and Computing Science, TUE. 1999-15

K.M.M. de Leeuw. *Cryptology and Statecraft in the Dutch Republic*. Faculty of Mathematics and Computer Science, UvA. 2000-01

T.E.J. Vos. *UNITY in Diversity. A stratified approach to the verification of distributed algorithms*. Faculty of Mathematics and Computer Science, UU. 2000-02

W. Mallon. *Theories and Tools for the Design of Delay-Insensitive Communicating Processes*. Faculty of Mathematics and Natural Sciences, RUG. 2000-03

W.O.D. Griffioen. *Studies in Computer Aided Verification of Protocols*. Faculty of Science, KUN. 2000-04

P.H.F.M. Verhoeven. *The Design of the MathSpad Editor*. Faculty of Mathematics and Computing Science, TUE. 2000-05

J. Fey. *Design of a Fruit Juice Blending and Packaging Plant*. Faculty of Mechanical Engineering, TUE. 2000-06

M. Franssen. *Cocktail: A Tool for Deriving Correct Programs*. Faculty of Mathematics and Computing Science, TUE. 2000-07

P.A. Olivier. *A Framework for Debugging Heterogeneous Applications*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2000-08

E. Saaman. *Another Formal Specification Language*. Faculty of Mathematics and Natural Sciences, RUG. 2000-10

M. Jelasity. *The Shape of Evolutionary Search Discovering and Representing Search Space Structure*. Faculty of Mathematics and Natural Sciences, UL. 2001-01

R. Ahn. *Agents, Objects and Events a computational approach to knowledge, observation and communication*. Faculty of Mathematics and Computing Science, TU/e. 2001-02

M. Huisman. *Reasoning about Java programs in higher order logic using PVS and Isabelle*. Faculty of Science, KUN. 2001-03

I.M.M.J. Reymen. *Improving Design Processes through Structured Reflection*. Faculty of Mathematics and Computing Science, TU/e. 2001-04

S.C.C. Blom. *Term Graph Rewriting: syntax and semantics*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2001-05

R. van Liere. *Studies in Interactive Visualization*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2001-06

A.G. Engels. *Languages for Analysis and Testing of Event Sequences*. Faculty of Mathematics and Computing Science, TU/e. 2001-07

J. Hage. *Structural Aspects of Switching Classes*. Faculty of Mathematics and Natural Sciences, UL. 2001-08

M.H. Lamers. *Neural Networks for Analysis of Data in Environmental Epidemiology: A Case-study into Acute Effects of Air Pollution Episodes*. Faculty of Mathematics and Natural Sciences, UL. 2001-09

T.C. Ruys. *Towards Effective Model Checking*. Faculty of Computer Science, UT. 2001-10

D. Chkliaev. *Mechanical verification of concurrency control and recovery protocols*. Faculty of Mathematics and Computing Science, TU/e. 2001-11

M.D. Oostdijk. *Generation and presentation of formal mathematical documents*. Faculty of Mathematics and Computing Science, TU/e. 2001-12

A.T. Hofkamp. *Reactive machine control: A simulation approach using χ*. Faculty of Mechanical Engineering, TU/e. 2001-13

D. Bošnački. *Enhancing state space reduction techniques for model checking*. Faculty of Mathematics and Computing Science, TU/e. 2001-14

M.C. van Wezel. *Neural Networks for Intelligent Data Analysis: theoretical and experimental aspects*. Faculty of Mathematics and Natural Sciences, UL. 2002-01

V. Bos and J.J.T. Kleijn. *Formal Specification and Analysis of Industrial Systems*. Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2002-02

T. Kuipers. *Techniques for Understanding Legacy Software Systems*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2002-03

S.P. Luttik. *Choice Quantification in Process Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-04

R.J. Willemen. *School Timetable Construction: Algorithms and Complexity*. Faculty of Mathematics and Computer Science, TU/e. 2002-05

M.I.A. Stoelinga. *Alea Jacta Est: Verification of Probabilistic, Real-time and Parametric Systems*. Faculty of Science, Mathematics and Computer Science, KUN. 2002-06

N. van Vugt. *Models of Molecular Computing*. Faculty of Mathematics and Natural Sciences, UL. 2002-07

A. Fehnker. *Citius, Vilius, Melius: Guiding and Cost-Optimality in Model Checking of Timed and Hybrid Systems*. Faculty of Science, Mathematics and Computer Science, KUN. 2002-08

R. van Stee. *On-line Scheduling and Bin Packing*. Faculty of Mathematics and Natural Sciences, UL. 2002-09

D. Tauritz. *Adaptive Information Filtering: Concepts and Algorithms*. Faculty of Mathematics and Natural Sciences, UL. 2002-10

M.B. van der Zwaag. *Models and Logics for Process Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-11

J.I. den Hartog. *Probabilistic Extensions of Semantical Models*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2002-12

L. Moonen. *Exploring Software Systems*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-13

J.I. van Hemert. *Applying Evolutionary Computation to Constraint Satisfaction and Data Mining*. Faculty of Mathematics and Natural Sciences, UL. 2002-14

S. Andova. *Probabilistic Process Algebra*. Faculty of Mathematics and Computer Science, TU/e. 2002-15

Y.S. Usenko. *Linearization in μCRL*. Faculty of Mathematics and Computer Science, TU/e. 2002-16

J.J.D. Aerts. *Random Redundant Storage for Video on Demand*. Faculty of Mathematics and Computer Science, TU/e. 2003-01