Modelling and Verification of Real-Time Systems Using Timed Automata: Theory and Practice

Paul Pettersson

A Dissertation submitted
for the Degree of Doctor of Philosophy
Department of Computer Systems
Uppsala University

February 1999

DoCS 99/101

ISSN 0283-0574

Dissertation for the Degree of Doctor of Philosophy in Computer Systems
presented at Uppsala University in 1999.

ABSTRACT

Pettersson, P. 1999: Modelling and Verification of Real-Time Systems Using Timed Automata: Theory and Practice. DoCS 99/101. 206 pp. Uppsala. ISSN 0283-0574.

During the last decade, model-checking techniques for the verification of timed system have been developed based on the theory of timed automata. The practical limitation in applying these techniques to industrial-size systems is the huge amount of time and memory needed to explore and store the state-space of the system model.

In this thesis, we improve the current status of model-checking techniques for timed systems by developing symbolic, on-the-fly and compositional verification techniques for timed automata. A common characteristics of the model-checking techniques presented is that they use efficient constraint-solving techniques to symbolically represent and manipulate the state-space. To avoid construction of the full state-space of the system model two techniques are used: on-the-fly generation of the state-space and a compositional model-checking technique. The memory-usage is further reduced by developing a minimal and canonical data structure for the class of constraints used in the model-checking algorithm, which reduces the size of each individual state. Two other techniques to reduce the total number of states explored and stored during verification are also presented.

The developed techniques have been implemented in the verification tool UPPAAL. To demonstrate the potential applications of our model-checking techniques, we present three industrial-size case studies where the UPPAAL tool is applied.

Paul Pettersson, Department of Computer Systems, Uppsala University, Box 325, S-751 05 Uppsala, Sweden, Email: paupet@docs.uu.se

ISSN 0283-0574

Printed in Sweden by Nina Tryckeri HB, Uppsala 1999.
Distributor: Department of Computer Systems, Uppsala University, Box 325,
S-751 05 Uppsala, Sweden.

Till Sara, Egil och Frej


ARTES
Real-Time PhD
http://www.docs.uu.se/artes

Acknowledgements

This thesis would not have been possible without the advice and support of my supervisor and friend Wang Yi. He has, during my five years as graduate student, guided me with enthusiasm and great humour, and supported me to grow as a researcher. I am very grateful for his thorough reviewing of this thesis and our collaboration that has resulted in a number of publications, of which several are included in this thesis.

I am also grateful to my other colleagues in the "UPPAAL group" at the Department of Computers System (DoCS) at Uppsala University, i.e. Tobias Amnell, Johan Bengtsson, Alexandre David, Fredrik Larsson and Justin Pearson for the stimulating environment. In particular, I would like to thank Johan Bengtsson and Fredrik Larsson for their excellent work in implementing the UPPAAL tool over the years and many fruitful discussions.

I would like to thank Bengt Jonsson and Kim G. Larsen for their continuous support over the years. I have benefitted a lot from their insights in theoretical computer science. In particular, I thank Kim for co-authoring several papers included in this thesis, and other publications not included. I thank him also for giving me the opportunity to visit the Department of Computer Science at Aalborg University in Denmark during ten weeks in 1995.

I acknowledge my other co-authors, i.e. Mats Daniels. W. O. David Griffioen, Kåre J. Kristoffersen, Francois Laroussinie, Magnus Lindahl and Henrik Lönn. Thank you for your cooperativeness and for the fun we had in writing the papers and presenting them at conferences.

Of all my friends and colleagues at DoCS, I like to thank in particular Hans Hansson for his support and leadership in the ASTEC-RT project, the secretaries Helena Pettersson and Inga-Lisa Ericsson, and the system administrators Per Lindgren and Anders Andersson for all the help I received over the years.

I would like to thank Tiziana Margaria for her constructive comments on an early version of this thesis.

To my fiance Sara Magnusson I can only say thank you. She has sacrificed a lot for this thesis, especially during the last months; she has my deepest gratitude and love. Finally, I thank my two sons Egil and Frej who are my best sources of inspiration and motivation. I am sorry that we have not been able to play much lately, I will probably have more time now...

This work has been partially supported by the Swedish Board for Technical Development (NUTEK), the Swedish Technical Research Council (TFR) and the competence center for Advanced Software TEChnology (ASTEC) at Uppsala University.

This thesis is based on seven revised versions of papers published in 1994 to 1998. In the introduction the papers will be referred to as papers A through G.

[A] Wang Yi, Paul Pettersson, and Mats Daniels. Automatic Verification of Real-Time Communicating Systems By Constraint-Solving. In Dieter Hogrefe and Stefan Leue, editors, *Proc. of the 7*th *Int. Conf. on Formal Description Techniques*, pages 223–238. North-Holland 1994.

[B] Kim G. Larsen, Paul Pettersson, and Wang Yi. Compositional and Symbolic Model-Checking of Real-Time Systems. In *Proc. of the 16*th *Real-Time Systems Symposium*, pages 76–87. IEEE Computer Society Press 1995.

[C] Kim G. Larsen, Paul Pettersson, and Wang Yi. Diagnostic Model-Checking for Real-Time Systems. In Rajeev Alur, Thomas A. Henzinger and Eduardo D. Sontag, editors, *Proc. of Workshop on Verification and Control of Hybrid Systems III*, number 1066 in Lecture Notes in Computer Science, pages 575–586. Springer–Verlag 1995.

[D] Fredrik Larsson, Kim G. Larsen, Paul Pettersson, and Wang Yi. Efficient Verification of Real-Time Systems: Compact Data Structures and State-Space Reduction. In *Proc. of the 18*th *Real-Time Systems Symposium*, pages 14–24. IEEE Computer Society Press 1997.

[E] Johan Bengtsson, W. O. David Griffioen, Kåre J. Kristoffersen, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. Verification of an Audio Protocol with Bus Collision Using UPPAAL. In Rajeev Alur and Thomas A. Henzinger, editors, *Proc. of 9*th *Int. Conf. on Computer Aided Verification*, number 1102 in Lecture Notes in Computer Science, pages 244–256. Springer–Verlag 1996.

[F] Henrik Lönn and Paul Pettersson. Formal Verification of a TDMA Protocol Startup Mechanism. In *Proc. of the Pacific Rim International Symposium on Fault-Tolerant Systems*, pages 235–242. IEEE Computer Society Press 1997.

[G] Magnus Lindahl, Paul Pettersson, and Wang Yi. Formal Design and Analysis of a Gear-Box Controller. In Bernhard Steffen, editor, *Proc. of the 4*th *Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, number 1384 in Lecture Notes in Computer Science, pages 281–297. Springer–Verlag 1998. Invited for publication in *International Journal on Software Tools for Technology Transfer*.

Comments on My Participation

All the papers included in this thesis are revised by me and Wang Yi.

Paper A: I participated in discussions and made the prototype implementation. All the proofs are by me and Wang Yi.

Paper B: I participated in discussions and wrote the section on experiments included in the conference version of the paper.

Paper C: I participated in discussions and wrote a large part of the paper. The case study is due to me.

Paper D: I participated in discussions and made the experiments. I wrote part of the paper.

Paper E: I participated in discussions. W. O. David Griffioen and I wrote sections 4 to 6.

Paper F: Henrik Lönn and I modelled the protocol. I wrote part of the paper.

Paper G: Magnus Lindahl and I modelled and synthesised the model of the gearbox controller and wrote sections 4 to 6.

Contents

Part I
Introduction

1 Background

In this thesis, I shall present the past five-years of research behind the following quotation:

> *"In 1996, Bengtsson and his colleagues model checked the entire protocol, thus completing the quest of fully automating a human proof that as little as two years ago was considered far out of reach for algorithmic methods."*
>
> *E.M. Clarke and J.M. Wing, Formal Methods: State of the Art and Future Directions, ACM Computing Surveys, Vol. 28, No. 4, 1996, page 631.*

First, we notice that *"Bengtsson and his colleagues"* refers to paper E included in this thesis. It reports an automated analysis of an audio-control *protocol* developed by Philips. The protocol is used in Philips audio and TV equipments for transferring control-information between components. For example, an amplifier can use the protocol to transfer information about which buttons are being pressed on the remote control to a connected CD-player.

The protocol by Philips is just one example of how information technology, i.e. the use of computers to create, store, exchange, and use information, has become a part of our everyday life. In fact, we trust computers to control much more safety-critical equipments than just audio and TV sets. For example, we use computer based systems to control anti-lock braking systems in cars, railway switching systems, banking systems, complex production processes, nuclear power plants, and military systems such as missiles [BW90, But97]. In most of these applications computer failures can lead to economical damage, environmental catastrophes, and in some cases, loss of human lives.

Over the past decades, it has been a challenge for computer scientists to develop theories and techniques that guarantee that computer systems operate correctly, i.e. according to prescribed specifications expressing their desired behaviour. The traditional ways of obtaining such "guarantees" have been simulation and testing. However, in many applications this process is exceedingly time-consuming and often provides only probabilistic measures of correctness.

In the literature, a great number of mathematically based techniques for reasoning about the correctness of computer systems have been proposed [Hoa69, Dij75, Pnu77, Lam77, Hoa78, Mil89, Hol91]. The general idea is to describe the computer system under consideration in a formal framework, and then apply rigorous methods to *prove* that the system description is correct in the sense that it satisfies certain formally specified requirements. The advantage of this approach is that it can be used early in

the design cycle to detect logical design bugs even before they have been implemented. However, the major drawback is that large formal system descriptions often tend to become complex and are therefore generally considered difficult to analyse [WT94].

To overcome this problem, techniques have been sought to analyse formal system descriptions automatically. One of the most promising approaches has been *model checking* [CE81, QS82, BCM+90, ACD90, Hol91, CW96]. In contrast to manual techniques, model checking is completely automatic in the sense that the proof showing that a system satisfies a given requirement is automatically generated.

In this thesis we shall study and develop model-checking techniques and tools for a special class of computer systems known as *real-time systems*. The audio-control protocol by Philips mentioned earlier is a typical example of such a system.

1.1 Real-Time Systems

A *real-time system* is a computer system whose correctness depends not only on the output, but also the time at which the output is produced. In fact, the examples we have mentioned so far are all real-time systems according to this definition. Consider for example an anti-lock brake system in a modern vehicle. Sensors provide it with information of the current wheel speed and it must react in a timely fashion when the driver applies the brake. If it does not react timely, it is not correct and will probably be of more harm than help to the driver.

The class of real-time system includes many *embedded* and *safety-critical system* that are subcomponents of a larger complex system operating in safety-critical environment [Sto96]. These systems are often known as *hard* real-time as they must always react timely, as opposed to *soft* real-time systems that may occasionally fail to meet their timing requirements [TH96]. A common characteristics of real-time systems is that they may consist of many components operating in parallel; they are then known as *concurrent systems*. As real-time system must react to every stimuli from the environment, they are also called *reactive systems* [Pnu86].

The term "real-time" is sometimes used for systems that react to external inputs as *quickly* as possible. Our definition requires the reaction to be *timely* in the sense that the system should react according to timing constraints. To reason about these systems we therefore need to define the meaning of precise timing more carefully. We shall assume a global time scale that is a time reference for both the system and its environment. A time scale indicates that time can be measured. However it does not imply a global clock in the system. The components of a real-time system may have their own clocks. Conceptually, these clocks can be considered as the local clocks of the components, which can be tested and reset. The components may communicate via channels. However, we shall only consider a simple form of synchronisation, namely handshaking that has been implemented by the rendezvous mechanism in many programming languages for real-time systems e.g. Occam 2 and Ada [Bar94], and in formalisms such as CSP [Hoa78] and CCS [Mil89].

This thesis will be focused on developing efficient techniques for analysing the behaviours of real-time systems. We first introduce a well developed model that we shall use to describe such systems.

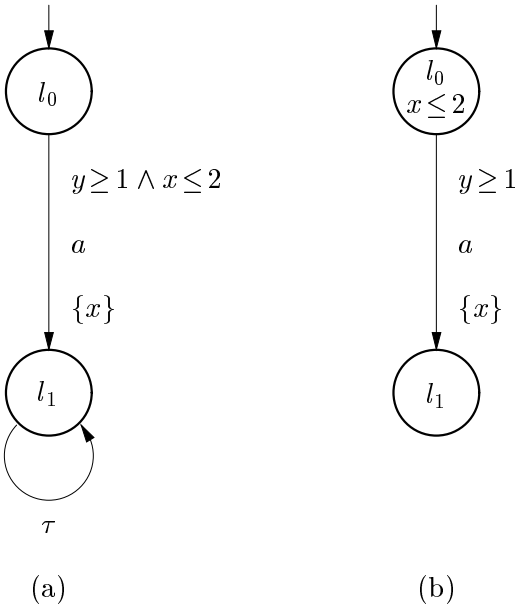


Figure 1: Two Timed Automata.

1.2 Timed Automata

There have been many formalism description techniques introduced in the literature for describing time-constrained systems [DS89, NRJV90, Yi91, Han91, AH92, AL92, BD91, Rok93, AD90, ACD90, Dil89, Hen91, Nan92, ACHH93]. One of the most successful formalisms is the model of timed automata [AD90, ACD90, HNSY92, NSY92a].

The model of timed automata was first introduced in 1990 by Alur and Dill [AD90] as an automata-theoretic approach for describing and analysing the behaviour of finite-state systems with real-valued clocks. In the following we outline the two first proposed models, timed graphs [ACD90] and timed Büchi automata [AD90], and compare them with a later proposed variant, called timed safety automata [HNSY92]. We also briefly introduce the model, networks of timed automata, which we shall work with in this thesis.

Timed Graphs

The model of *timed graphs* extends the model of finite-state automata with a finite collection of real-valued clocks [ACD90]. The clocks proceed synchronously (i.e. at the same rate) and measure the time since they were last reset. Syntactically, the edges of timed graphs are labeled with action names as in finite-state automata, but extended with clock constraints to test the clock values, and reset sets indicating the assignment of the clock values to zero. As an example consider the timed graph shown in Figure 1(a). It has two clocks, x and y, two *control locations* l_0 and l_1, and an *edge* from control location l_0 to l_1 labeled with the *guard* $y \geq 1 \wedge x \leq 2$, the action name a, and the *reset set* $\{x\}$.

A *state* of a timed graph is in the form (l, v), where l is a control location and v is an assignment mapping clock variables to non-negative real numbers. To introduce the semantics of timed graphs we reconsider the timed graph shown in Figure 1(a). Assuming that all clock variables are initially set to zero and the initial control location is l_0, the graph starts in the state $(l_0, \{x = y = 0\})$. As the clocks increase synchronously with time, it may evolve to all states of the form $(l_0, \{x = y = t\})$, where t is a non-negative real number. At any state with $t \in [1, 2]$ it may change to state $(l_1, \{x = 0, y = t\})$ by following the edge from l_0 to l_1, that resets x. However, it can also idle forever in location l_0 since there is no way of forcing progress in the semantics of timed graphs. Note that in the literature the notion of timed graphs is often known as timed automata.

Timed Büchi Automata

A Büchi automaton [Büc62] is a finite-state automaton with a subset of its states being accepting states. Only transition sequences with infinitely many accepting states are considered valid (i.e. accepting) runs of a Büchi automaton.

Timed Büchi automata [AD90] are Büchi automata extended with clocks in the same way as in timed graphs, but timed Büchi automata accept infinite runs only, in the above sense. As an example, we reconsider the timed automaton shown in Figure 1(a), but now interpreted semantically as a timed Büchi automaton. Assuming l_1 is the only accepting location in the automaton, all valid runs must pass through location l_1 infinitely often. Note that this implies that the automaton, in all valid runs, leaves location l_0 within 2 time units (but not before 1 time unit), as the guard $x \leq 2$ is false at any later time point. Thus, in the example and in general, the Büchi acceptance condition imposes implicit progress conditions on the control locations, since only the set of valid runs is considered when the behaviour of an automaton is analysed.

Timed Safety Automata

In contrast to the Büchi accepting condition which is a theoretically elegant way to impose progress conditions on locations in timed automata, there have also been other suggestions to specify progress conditions [HNSY92, NSY92a]. *Timed safety automata*, due to Henzinger et.al. [HNSY92], are timed automata without the acceptance condition. Instead, each location is labeled explicitly with a local progress condition in the form of a clock constraint, called a *location invariant*. Semantically, the location invariant is required to always hold when the automaton operates in the location and consequently the automaton must change location while the location invariant still holds. As an example, consider the timed safety automaton shown in Figure 1(b). It is similar to the timed Büchi automaton shown in Figure 1(a) in the sense that it must advance from location l_0 within 2 time units according to the location invariant.

We believe that the notion of local progress conditions in the form of location invariants is more appealing for modelling and automated analysis than the more

complex Büchi accepting condition. We shall therefore use timed automata with location invariants to model real-time systems.

Networks of Timed Automata

The systems we are interested in often consists of several communicating components executing in parallel. The problem of how to describe such systems is not explicitly addressed in the early work on timed automata [AD90, ACD90, HNSY92]. In [AD90], the parallel composition of timed automata is interpreted as logical conjunction, which is similar to the strong (multi-) synchronisation operator from process algebras, defined by the rule:

$$\frac{A \overset{a}{\Leftrightarrow\!\!\rightarrow} A' \qquad B \overset{a}{\Leftrightarrow\!\!\rightarrow} B'}{A \;\&\; B \overset{a}{\Leftrightarrow\!\!\rightarrow} A' \;\&\; B'}$$

Intuitively, it means that the whole system described by $A \;\&\; B$ may make a move (i.e. do an a) only if the components described by A and B can do the same (and at the same time). That is, all components of a composed system must synchronise on every action at every time point.

This seems to be too strong a restriction for applying timed automata in practice. Therefore, we shall instead work with networks of timed automata. A *network of timed automata* is simply the parallel composition $A_1|\cdots|A_n$ of a finite collection $A_1, \ldots, A_n$ of timed automata for a given synchronisation function $|$. In particular, we will use networks of timed automata composed with a CCS-like parallel composition operator [Mil89]. It allows for individual components to perform internal actions (i.e. interleaving), and for pairs of components to synchronise on actions.

1.3 Automatic Verification

The act of *verification* is that of using mathematically justified methods to prove that a formal system description satisfies certain desirable properties. In this thesis we are mainly concerned with *algorithmic* verification techniques where the property of interest is checked automatically, as opposed to techniques that require manual assistance. In particular, we shall develop *model-checking* algorithms for timed systems. Such algorithms determine if a desired property, formalised as a logical formula, is satisfied by a system model.

Safety Properties

To specify and reason about the correctness of untimed computer programs, logics have been very successful, e.g. pre- and postcondition pairs formulated in Hoare's logic have been used for sequential programs [Hoa69], and temporal logics for concurrent systems [Pnu77]. In recent years, several timed variants of these logics have also been developed for specifying real-time properties, see e.g. [AH92] for an overview.

In [Lam77], Lamport classifies temporal properties into *safety* and *liveness* properties. Intuitively, a liveness property asserts that "something good" eventually will happen, and a safety property that "something bad" should never happen [Lam80].

In this thesis, we shall restrict ourselves to studying safety properties only. This means that there are some properties that we will not be able to specify and verify[1]. However, there are several arguments in favour of the restriction in the context of real-time systems.

First, real-time systems are often embedded in safety-critical environments. It has therefore been argued that the practical goal of verifying real-time systems is often to show that "something bad" never happens [Hal93]. That is, to ensure that the system is "safe", which can be expressed as a set of safety properties.

Secondly, it is possible to formulate a form of liveness, known as *bounded liveness*, with safety properties according to Lamport's classification [Lam77]. Intuitively, a bounded-liveness property asserts that "something good" happens within a specified time bound. As we are concerned with systems that are required to operate under timing-constraints, it is even more interesting to reason about bounded liveness than unbounded liveness, e.g. some event will eventually occur in the future.

Finally, in choosing specification languages and verification algorithms there is a trade-off between *expressiveness*, i.e. which properties can be specified, and *complexity*, i.e. how difficult properties are to verify [Hen91]. In general it is possible to find more efficient algorithms for logics with restricted expressiveness. Safety properties can be verified by reachability analysis algorithms that examine all possible states of a system to ensure that "something bad" never happens. This has been used to develop efficient algorithms for finite automata. For timed automata however, the state-space is infinite because of the real-valued clocks; but it is possible to represent the infinite state-space by finite partitioning due to the region-graph construction technique [AD90].

Region-Graph Construction

In the first papers about timed automata it is shown that fundamental verification problems associated with the model, such as language emptiness and model-checking of timed temporal logic formulae, are decidable [AD90, ACD90]. The result is obtained by constructing a finite-state system called a *region graph*. In a region graph, each region essentially consists of the set of concrete states that are equivalent, in the sense that they can evolve to the same regions in the future.

Based on Alur and Dill's pioneer work, several other algorithms were developed for the verification problems associated with the model, e.g. [ACD90, Nan92, Cer92]. However, in practice these algorithms are often computationally infeasible because of the huge size of the region graph. The region graph grows exponentially, not only in the number of automata and the number of clock variables, but also in the largest integer constant used in the clock constraints of the automata.

Symbolic Verification Algorithms

Subsequently, techniques have been sought to develop algorithms that perform better in practice, in the sense that they run faster and consume less space on the instances

[1] For example, we will not be able to reason about zeno-properties [AL92].

of timed automata that appear when realistic systems are modelled. One way to develop such algorithms have been to use coarser and consequently more compact abstract representations of the state-space [Dil89, HNSY92, ACHH93, Hal93]. We shall call such algorithms *symbolic* since the term has been used in the literature on verification of timed systems[2]. It should be noticed that the region-graph technique is also a symbolic technique.

The first progress in the development of symbolic model-checking algorithms for timed automata is presented in [HNSY92]. A symbolic model-checking algorithm is described that partitions the concrete state-space into equivalent classes, in such a way that states within each class are equivalent with respect to the investigated automaton and the property currently being checked. Though the algorithm in the worst case will construct (the equivalent of) the full region-graph, it often yields a coarser and smaller partitioning. In practice, the algorithm is insensitive to the maximal integer constant appearing in the clock constraints of the automata and the checked property. However, it is still very sensitive to the number of automata in the analysed system as the product of the automata is constructed before verification is performed.

On-the-Fly State-Space Generation

An *on-the-fly* verification algorithm constructs the state-space of the investigated system and verifies the property of interest simultaneously, in contrast to the traditional approach of generating the whole state-space before verification. This allows the verification to be stopped when the truth-hood of the verified property has been determined. On-the-fly techniques therefore often require a relatively small part of the whole state-space to be investigated.

Several on-the-fly verification algorithms have been developed for finite-state systems in the literature, e.g. [VW86, Hol91]. In this thesis, we shall investigate how on-the-fly techniques can be adopted to verification algorithms for real-time systems to deal with the exponential growth of the state-space caused by the number of automata in parallel systems. Moreover, we shall investigate the possibility of combining on-the-fly techniques with symbolic verification techniques to develop algorithms that are less sensitive to the number of automata and insensitive to the constants appearing in the clock constraints of the analysed automata.

Compositional Verification

The notion of *compositional verification* in general relates to methods for splitting up the verification of a combined system into verifications of its components. This means that a compositional verification technique shows that a system satisfies properties without investigating the product of the whole system, but rather its components one by one.

[2] In the context of verification techniques for finite-state systems, the term "symbolic" is often used for non-enumerative methods of state-space representation and exploration, such as Binary Decision Diagrams (BDD's) [Bry86, BCM+90].

For untimed systems a number of compositional verification techniques have been described in the literature, e.g. [OG76, Sti86, Lar86, Jon87, And95]. In this thesis, we shall develop compositional verification algorithms for real-time systems, based on a compositional model-checking technique for untimed concurrent systems, known as *quotienting*, introduced by Larsen [Lar86] and further studied by Andersen [And95]. The technique avoids exploring the whole state-space of a composed system by gradually moving components from the system description to the specification.

An initial attempt to transfer Andersen's result to a real-time setting is reported in [LL95]. This work also contains some experimental evidence showing the potential of the quotienting technique for real-time systems. However, the result in [LL95] is obtained by using the region-graph technique.

2 Summary of Results

In this section we summarise the main contributions of this thesis. The presentation is divided into four areas: symbolic verification algorithms, memory usage reductions, verification tool developments, and case studies.

2.1 Symbolic Verification Algorithms

On-the-Fly Verification (Papers A and B)

A reachability analysis algorithm for verifying safety properties of networks of timed automata is developed in paper A. In contrast to previously proposed algorithms for timed automata, ours operates in an on-the-fly manner. That is, it verifies properties during state-space exploration, instead of constructing and representing the whole state-space before the checking. The algorithm is also symbolic in the sense that the infinite state-space is finitely partitioned into subsets that are represented and manipulated using a class of linear constraints, known as *difference bound matrices* (DBM) [Bel57, Dil89]. To develop the algorithm, we give a finite symbolic semantics of timed automata, which is defined as a transition system with transition rules given in terms of predicates and operations on constraints. This means that we reduce the verification problem of timed automata to that of manipulating and solving simple constraints.

In paper B these results are applied to develop a symbolic and on-the-fly model-checking technique for checking a simple timed modal logic $\mathcal{L}_s$ to express safety and bounded liveness properties. The logic can be seen as a fragment of the timed modal μ-calculus (T_μ) [HNSY92]. Though it is comparatively less powerful than e.g. T_μ and TCTL [ACD90], our logic is still sufficiently powerful for practical purposes; a claim we substantiate by showing that a number of operators of other real-time logics can be expressed as derived operators in $\mathcal{L}_s$. Most importantly, the somewhat restrictive expressive power of $\mathcal{L}_s$ allows us to apply on-the-fly and constraint-solving techniques to develop an efficient model-checking algorithm for timed automata.

Compositional Verification (Paper B)

In paper B, we present a symbolic and compositional verification technique for checking $\mathcal{L}_s$-formulae of networks of timed automata. It extends quotienting for (untimed) finite-state systems [And95] to real-time systems where subsets of the infinite state-space are manipulated symbolically using constraint-solving techniques. This means that the presented technique confines both the exponential blow-up in the state-space caused by the number of components, and the exponential growth caused by the number of clocks.

Like the quotienting technique for finite-state systems [Lar86, And95], ours solves the model-checking problem without exploring or even generating the state-space of the automata network. Instead, timed automata are gradually moved from the system description to the requirement specification. More precisely, given the model-checking problem $(A|B) \models \varphi$, where $(A|B)$ is the system to be verified, with the two timed automata A and B, and φ is a $\mathcal{L}_s$-formula, the quotienting technique shows how to construct the *quotienting formula* φ/B such that

$$(A|B) \models \varphi \text{ if and only if } A \models (\varphi/B)$$

Repeating this processes yields the equivalent model-checking problem $\models ((\varphi/B)/A)$. Thus, roughly speaking the quotienting technique transforms the model-checking problem to that of checking the truth-hood of the quotienting formula $((\varphi/B)/A)$.

2.2 Memory Usage Reduction

In papers D and E, we present three techniques to reduce the memory usage of real-time verification algorithms. Note that the techniques are orthogonal and can thus be applied in combination.

Control Structure Reduction (Paper D)

An on-the-fly technique to reduce the space-consumption of reachability analysis algorithms for timed automata is described in paper D. The technique is based on the observation that not all symbolic states encountered during state-space exploration, but only certain critical symbolic states, need to be saved to ensure termination. Before reachability analysis, the control structure of the automata in the network are statically analysed. Based on this information we are able to compute, in an on-the-fly manner, a set of symbolic states to save, that is sufficient for guaranteeing termination of the reachability analysis algorithm. The set of saved symbolic states may not be minimal but the reduction technique performs well in practice. In an experiment with an implementation of the control structure reduction technique in the tool UPPAAL we found that for six examples from the literature, the space-saving is between 13% and 72%.

Compact Data Structures for Constraints (Paper D)

In paper D we also present a compact data structure for difference bounded matrices (DBM) [Bel57, Dil89, BL96], the class of constraints that arises during symbolic

verification of timed automata. The data structure is based on an $\mathcal{O}(n^3)$ reduction algorithm that, given a constraint system with n clocks, constructs an equivalent reduced system with the *minimal* number of constraints. The reduced system is *canonical* in the sense that two constraint systems with the same solution sets reduce to identical constraint systems. The reduction algorithm is essentially a minimisation algorithm for weighted directed graphs, which extends the transitive reduction algorithm of [AGU72] to weighted graphs.

The space usage of a reduced constraint system is in the same order as for an ordinary DBM, i.e. $\mathcal{O}(n^2)$, but it often turns out to be much less in practice. We have tested compact data structures for constraints in an experiment with six examples from the literature. The number of constraints was reduced with 68% to 85%.

Committed Locations (Paper E)

In paper E we develop the notion of *committed locations*. It allows for atomic behaviours, such as atomic broadcasts, to be accurately modelled in the model of networks of timed automata. More importantly, committed locations are utilised to guide the state-space exploration performed during verification by on-the-fly reachability analysis, to avoid exploring unnecessary interleavings of independent transitions. We present a modified symbolic and on-the-fly reachability analysis algorithm for networks of timed automata which explores and stores a reduced number of symbolic states when committed locations are used. Our experimental results demonstrate significant time and space-savings of the modified model-checking algorithm when committed locations are used.

2.3 Verification Tool Development

The two verification tools TAB and UPPAAL have been developed based on the techniques and algorithms presented in this thesis.

The TAB Tool (Paper A)

TAB is the first prototype of UPPAAL developed in 1993 at Uppsala University. It is written in Prolog and the general constraint solver PCS (Prolog Constraint Solver) [Nil93]. The implementation is based on the symbolic and on-the-fly reachability analysis algorithm developed in this thesis. Although the TAB tool is a prototype, it has successfully been applied to automatically verify safety properties of some non-trivial examples, including a version of Fischer's mutual exclusion protocol and a railway control-system. These examples, as well as the theoretical foundations of the tool TAB are presented in paper A.

The UPPAAL Tool (Papers A, B, C, D and E)

UPPAAL (UPPsala and AALborg) is a suite of tools for modelling, validation and verification of real-time systems, developed in collaboration between Uppsala University and Aalborg University since 1995 [LPY97a, BLL+98]. The modelling language in

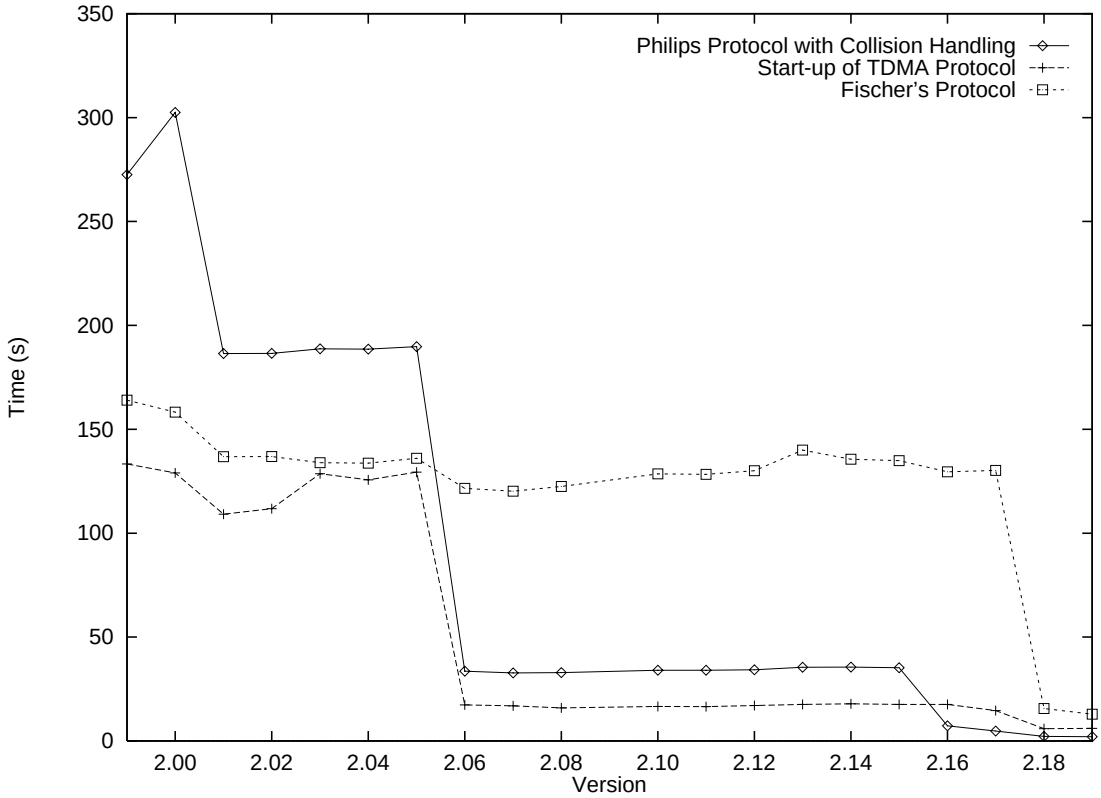


Figure 2: Time benchmarks (in seconds) for UPPAAL version 1.99–2.19. Version 1.99 and 2.19 are dated December 1996 and September 1998 respectively.

UPPAAL is networks of timed automata extended with data variables with finite domains, and arrays of such variables. The verification engine is based on the on-the-fly and symbolic constraint solving techniques presented in this thesis.

In contrast to its predecessor TAB in which all constraint solving is implemented in PCS, the constraint operations in UPPAAL are implemented entirely in C^{++} based on DBM [BL96]. To further improve the performance, the notion of committed locations as well as the control structure reduction and the compact data structures for constraints have been implemented. Moreover, a simple but efficient strategy for reuse is applied in the tool [LPY97b]. If possible, when several properties of a system are analysed, the generated portion of the state-space is reused, thus avoiding time-consuming re-computations.

Figure 2 and 3 illustrate how the time and space performance of UPPAAL have improved from version 1.99 to version 2.19 in terms of three examples[3]: a TDMA start-up algorithm, Fischer's mutual exclusion protocol with 5 processes and Philips audio-control protocol with bus collision. In particular, we notice that in both the time and space usage diagrams there is a performance improvement in version 2.06 compared with the proceeding version. This is due to a number of internal improve-

[3]All UPPAAL versions in the test are compiled using GCC 2.7.2.3 and installed on the same Pentium II 375 MHz machine running Redhat Linux 5.1.

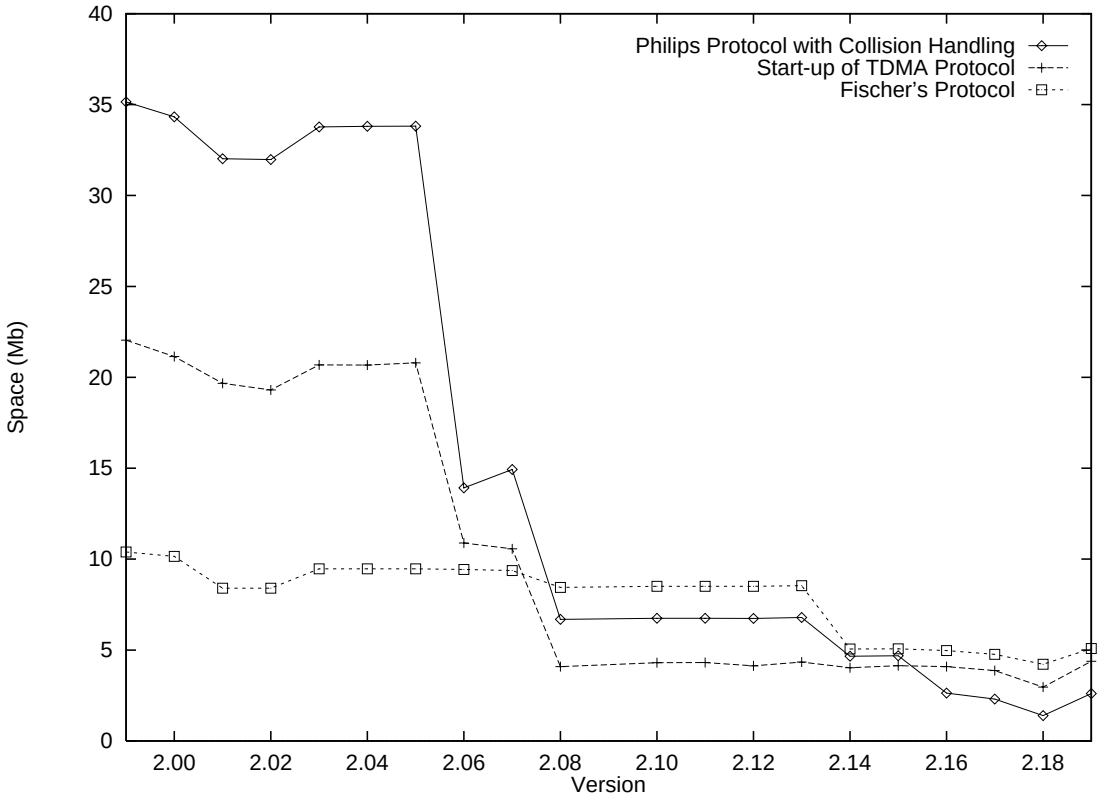


Figure 3: Space benchmarks (in Mb) for UPPAAL version 1.99–2.19.

ments in the verifier, including re-implementation of the main data structure for the explored symbolic state-space. The tool Purify played a significant role in these improvements.

Besides efficiency, the main design criterion of UPPAAL has been user-friendliness. To make debugging of system descriptions easier, the verifier has been extended with support for diagnostic traces (see paper C). When the verification of a particular property succeeds (or fails) an example is automatically produced that shows why the property is (or is not) satisfied by the analysed system model. The user-friendliness has been further improved by the development of graphical interfaces for the various components in the tool.

UPPAAL has been successfully applied in many case studies, to model, design and analyse complex real-time systems, see e.g. paper C, E, F and G. The theoretical foundations of UPPAAL are presented in the papers A, B, C, D and E.

2.4 Case Studies

Philips Audio-Control Protocol with Bus-Collision Handling (Paper E)

In paper E, we analyse a protocol developed by Philips for the physical layer of a bus protocol connecting the various devices, such as amplifiers and CD-players etc,

in audio equipment[4]. A formal description of the protocol is given in the model of timed automata. Four automata are used to model the protocol components and another three automata are used to encode assumptions about its environment and to test its behaviour. As main verification results, we prove using UPPAAL that the protocol model behaves correctly if the error on all timing in the components is bound to $\pm 5\%$, and incorrectly if the error is $\pm 6\%$. In addition, we model and analyse an erroneous version of the protocol, actually implemented by Philips. Using UPPAAL's ability to generate diagnostic traces when verification fails, we are also able to find and study an error scenario.

Start-Up Algorithm of a TDMA Protocol (Paper F)

The start-up algorithm of a time division multiple access (TDMA) based bus protocol is analysed in paper F. The protocol is used in DACAPO [RLST95], a conceptual computer architecture for safety-critical distributed real-time systems. It is intended for physically small distributed systems limited to tens of meters and less than 40 nodes, e.g. operating in modern vehicles. We formally describe the communicating nodes on the bus, and the bus itself, in the model of timed automata. To express correctness properties and assumptions about the protocol, two extra automata are introduced. Using UPPAAL we manage to show that a network of four communicating nodes is guaranteed to become synchronised and operational within a bounded time from an arbitrary initial state, given a clock-drift corresponding to $\pm 10^{-3}$ between the nodes. Furthermore, we derive an upper time bound for the start-up algorithm to complete, corresponding to 21 TDMA time slots, and describe a discovered worst-case scenario.

Gearbox Controller (Paper G)

The gearbox controller is a component in the real-time embedded system that operates in a modern vehicle. In paper G, we presents a case study where UPPAAL is applied to describe and analyse design sketches of a gearbox controller in the early design stages of a project. The controller receives gear-requests from the driver[5] via a communication network and implements the actual gear changes by controlling the involved lower level components, such as the gear, the clutch and the engine. We present a formal description of a gearbox controller, designed from informal requirements prescribed by our industrial partner, the Swedish company Mecel AB. The description, which comprises both a gearbox controller and abstract models of the components in the surrounding environment, is given as a network of five timed automata. We also formally describe Mecel's informal requirements as a set of 47 safety properties. Correctness of the modelled system is established using UPPAAL's simulator to validate the design, and the model-checker to verify that the formal requirements are satisfied.

[4] In paper C we analyse a smaller version of Philips audio-control protocol, without bus-collision handling.

[5] Gear changes may also be requested by a dedicated component implementing a gear change algorithm.

3 Related Work

During the past decade a large number of formalisms for real-time systems have been proposed in the literature, including timed process algebras [DS89, NRJV90, Yi91, Han91, DB96], timed Petri net models [Sif77, BD91, Rok93], real-time logics [AH92, AL92], duration calculus [CHR91], timed automata [AD90, ACD90, HNSY92, AHV93, SV96], and other state based approaches [Dil89, Hen91, Nan92, ACHH93]. In this section we shall only outline work related to timed automata.

3.1 Verification of Real-Time Systems

Several algorithms for checking properties of timed automata are described in [HNSY92, KL94, WT94, LL95, SS95, HKV96, Bal96, TC96, BMPY97, BTY97, AJ98, DT98].

In [HNSY92], Henzinger et.al. gives a symbolic model-checking algorithm which iteratively computes the set of (symbolic) states of a timed automata that satisfies a given T_μ-formula (or TCTL-formula). Whereas the symbolic representation of timing information used in the algorithm is similar as ours; it aims at a much more general framework in the sense that the logic T_μ is strictly more expressive than $\mathcal{L}_s$, the most expressive logic we consider. The restrictive expressive power of our logic allows for efficient model-checking, which has been demonstrated in experiments (see [BLL+95]). Though our logic is restricted it allows to specify invariant and bounded liveness properties that are sufficient for practical purposes.

Wong-Toi presents, in his thesis, a symbolic reachability analysis algorithm for timed automata [WT94]. Its distinguishing feature is to combine symbolic representation of both timing and control information with approximation techniques. To efficiently represent control information he uses *binary decision diagrams* (BDD) [Bry86]. Timing information is symbolically represented with constraints over clocks. Compared with our work, Wong-Toi focuses on finding useful over- and underapproximations techniques to compute super- and subsets of the symbolic state-space respectively. We consider exact verification techniques only.

In [Bal96] Wong-Toi's ideas are further developed by using BDD's also to symbolically represent timing information. The presented algorithm computes, in a more efficient way, an overapproximation of the symbolic state-space[6]. Hence, it can be applied to safely check that undesired combinations of the automata control locations and clock assignments are not reachable in a system model. However, since a superset of the symbolic state-space is computed, the negation can not be checked reliably, i.e. if states are reachable. Furthermore, it should be noticed that overapproximating the state-space can also be more expensive than computing the exact state-space, since in general a super-set of the control-nodes may be generated.

Another symbolic model-checking approach for timed automata based on BDD techniques is presented in [BMPY97]. Roughly speaking, the algorithm uses an extension of BDD's, called *numerical decision diagrams* (NDD's) [AMP97] to represent a discretization of the region graph [GPV94]. Therefore, the algorithm, like the size of the region graph, is very sensitive to the constants used in the clock constraints

[6] In fact, algorithms for computing two different overapproximations are presented.

of the analysed timed automata and logical formulae. However, it performs well on systems with small constants, e.g. some instances of Fischer's protocol [BMPY97].

Verification algorithms combining symbolic and on-the-fly techniques for timed automata are presented in [SS95, BTY97, DT98]. In [SS95], the specification language of consideration is an extension of the modal μ-calculus for real-time systems, called L^t_μ. Compared with our work, again, a much more expressive logic is considered. As a consequence, the algorithm needs to keep track of more information during verification. It must also gradually refine the symbolic state-space during operation by splitting states into finer partitions. This is not needed in our algorithms since we consider the more restrictive logic $\mathcal{L}_s$.

Another expressive logic, $\mathrm{TECTL}^*_\exists$, is considered in [BTY97]. The developed verification algorithm checks for emptiness [AD90, TC96] of the symbolic state-space of possibly several timed automata composed in parallel with the formula, expressed as a (negated) timed Büchi automaton. Emptiness is checked by a standard loop detection algorithm that explores the symbolic state-space of the timed automata using a depth-first strategy. Furthermore, the algorithm uses exactly the same representation of the symbolic state-space as we do. Thus, in comparison with our work, the main difference is that a loop detection algorithm is applied, instead of reachability analysis.

There have also been successful applications of the abstract interpretation framework [CC77] to develop symbolic verification algorithms for real-time systems. In [Hal93], Halbwachs presents an approximative algorithm for checking safety properties of time-constrained reactive systems based on linear relation analysis, an application of abstract interpretation. This work successfully demonstrates the power of the chosen approach, but it is focused on efficiently overapproximating the state-space.

Some results in the area of verification algorithms for Petri net models with time are in fact related to model-checking algorithms for timed automata. In [BD91] an algorithm is presented for verifying reachability, i.e. if a marked place in a safe Petri net is reachable or not. It is based on a symbolic semantics of time Petri nets and linear constraints over clock variables are used to represent timing information. The reachability analysis algorithm of Rokicki [Rok93] for his orbital nets also uses clock constraints. Notably, Rokicki also presents a number of cleverly optimised algorithms for handling clock constraints [Rok93, RM94]. These algorithms are directly applicable in verification algorithms for timed automata that represent clock constraints by difference bound matrices.

3.2 Tool Development

There have been a number of verification tools for real-time systems developed during the past few years, including: KRONOS, Rt-Spin, HYTECH, CMC, Epsilon [CGL93], Polka [HRP94], TREAT [KL96], RT-Cospan [AK95], SGM [WH98], and the implementations described in [WT94]. In the following we briefly describe the tools mostly related to this thesis and compare them with our tool UPPAAL.

The verification tool KRONOS [DOY94, DOTY95, BDM+98] is originally based on the symbolic model-checking algorithm for checking TCTL-formulae of timed

safety automata presented in [HNSY92, NSY92b]. The main verification algorithm in KRONOS therefore, in contrast to UPPAAL, generates the product of the analysed system before verification. It is also possible in KRONOS to check if timed automata are timed abstracting bisimilar [LY93, YT96]. Recently, the tool has also been extended with on-the-fly verification techniques to check safety properties of timed automata [BTY97].

Various algorithms to reduce the number of clock variables in a system of timed automata, without changing its behaviour, have been implemented in the separate tool OPTIKRON [DY96, Daw98]. It supports the same file format for timed automata as KRONOS, and can thus be used to improve the performance of KRONOS [7].

Tripakis and Courcoubetis extends the verification tool Spin and its modelling language Promela for real-time in [TC96]. In the resulting tool, called RT-Spin, timed Büchi automata are used both as a modelling language for describing real-time system, and as a specification language for formalising requirements. Thus, its users are required to specify properties as (negated) timed Büchi automata. The verification algorithm in RT-Spin is essentially an on-the-fly, depth-first search, loop detection algorithm that checks for language emptiness of the modelled system composed in parallel with the currently checked property. The symbolic semantics of timed automata and the constraint solving techniques adopted in the tool is exactly the same as in UPPAAL.

CMC [KLL+97, LL98] is a compositional model-checking tool for networks of timed automata based on the algorithms presented in [LL95] and further developed in paper B of this thesis. Its specification language is the logic L_ν[LL95, LL98] that might be seen as a fragment of the logic T_μ, but more powerful than $\mathcal{L}_s$. Besides compositional model-checking, it is also possible to check if two timed automata are bisimilar.

HYTECH (The HYbrid TECHnology Tool) [HHWT95, HH95, HHWT97] analyses properties of systems modelled as collections of linear hybrid automata [ACHH93], a strictly more expressive model than timed automata. The tool is therefore not directly comparable with the previously mentioned tools for the model of timed automata. However, when applied to real-time system, HYTECH allows for verification of reachability properties. It is also possible to instruct HYTECH to treat certain variables in the automata as parameters and compute a condition on the parameter under which the automata satisfy a given property [HHWT95].

3.3 Case Studies

Naturally, all the developed tools have been tested in case studies. The number of successful applications in the literature is large and expanding, e.g. [Gri94, WT94, Abr95, HWT95, KP95, DY95, HWT96, DKRT96, CW96, MY96, DKRT97, LSW97, HSLL97, CCMM97, SMF97, TY98, BFK+98, BFM98]. It is impossible to survey them all here. We therefore restrict to case studies closely related to the work of this thesis.

In [HWT95], Ho and Wong-Toi report a case study where HYTECH is used to automatically analyse a simplified version of the Philips audio-control protocol, pre-

[7] An extension of OPTIKRON for handling the UPPAAL file format is in work [Daw98].

viously studied without tool support in [BPV94]. The same version of the protocol was later analysed by other tools, e.g. UPPAAL (see paper C) and KRONOS [DY95]. The audio-control protocol we describe and analyse in paper E of this thesis is a more complete version of the protocol, which also includes bus-collision handling [LSW97, CW96]. It is considerably larger than the simplified version, in the sense that the number of control-nodes, components, and variables are increased.

A real-time protocol from the same application area, i.e. audio/video components, by Bang & Olufsen is analysed in [HSLL97]. Though it was known to be faulty, the error in the protocol could not be found using conventional testing methods. In the case study UPPAAL is used to produce an error-trace that reveals the error. UPPAAL is also applied to verify a suggested corrected version of the protocol, which is shown to satisfy the specified requirements.

Another industrial case study is described in [SMF97]. A control system by BMW AG which controls the height of a vehicle by regulating a pneumatic suspension system is modelled as a linear hybrid system. Correctness of the control system is formally specified as a number of safety properties and checked by symbolic reachability analysis in HYTECH.

D'Argenio et.al. presents a successful verification of a bounded retransmission file transfer protocol for lossy channels [DKRT96, DKRT97]. The protocol is based on the alternating bit protocol, but allows for a bounded number of retransmissions. To analyse all aspects of the protocol, two formal descriptions are given. An untimed, more data-oriented, version is modelled in Promela and analysed in the verification tool Spin [Hol91]. A timed version with quantified timeout values is modelled as a network of timed automata and checked to satisfy a number of safety properties, regarding the timing behaviour of the protocol, using UPPAAL.

In [KLL+97], the compositional model-checking tool CMC is applied to verify the correctness of Fischer's protocol [AL92]. The mutual exclusion property of a protocol instance with 50 processes is verified using only 173 seconds[8] and 32 MB of memory.

Many other time-constrained systems have been modelled as networks of timed automata and specified with safety properties. Examples hereof include: the mine pump problem presented and studied in [JBW+96], the steam boiler problem proposed in [Abr95] and further studied in [KP95], the collision avoidance protocol implemented on-top of an Ethernet-like medium analysed in [JLS96, ABL98], the lip synchronisation algorithm for synchronising multiple information streams sent over a communication network, described in [BFK+98], and the multimedia stream modelled and analysed in [BFM98].

4 Conclusion and Future Work

In this thesis, we have developed a number of model-checking techniques for timed systems, that have been implemented in the UPPAAL tool. To evaluate the practical applicability of these techniques and impact on tool development, we have presented several case studies. In this section, we discuss the potential implication of our results

[8] The experiment was performed on a machine running SunOS 5.5.

and lessons learnt from the work. We conclude the thesis with an outline of future work.

The main lesson learnt from our studies on verification techniques for timed systems is that relatively small restrictions on the expressive power of specification languages have rendered much more efficient verification techniques. In particular, the restriction to safety and bounded liveness properties has been of crucial importance to the efficiency of the UPPAAL model-checker. As the class of properties can be checked by reachability analysis, various optimisations on the model-checker have been focused on the on-the-fly symbolic reachability algorithm. For the compositional quotienting technique the restriction has also been important. By restricting to the safety and bounded liveness logic $\mathcal{L}_s$ it allows the development of efficient formulae simplification strategies, which has been important to the success of the quotienting technique implemented by Francois Laroussinie in the compositional model-checker CMC [KLL+97, LL98].

The presented verification techniques and data structures have all been implemented in the UPPAAL tool except the quotienting technique. Our experiments demonstrate that they have dramatically reduced the time- and space-requirements of verification algorithms in many practical applications. However, it is important to note that they can not in general improve the worst-case complexity of verification algorithms for timed automata. It is possible to find other examples where the situation is not much improved, or where other verification techniques perform better, see e.g. [BMPY97]. It is also important to note that the reduction techniques are completely independent and can thus be combined.

Most of the examples in our case studies have been taken from the literature. We believe in that they are of general interest as they have also been studied in other contexts and verification tools. We have also presented one industrial case study which is carried out jointly with Mecel AB, a company developing control systems for modern vehicles. In this study, we developed a prototype controller for a gearbox. The case study demonstrates the potential industrial applications of the model-checking technique. In particular, it shows that the model-checking technique may be used as a powerful debugging tool in the design process of real-time embedded systems.

There are many possible future extensions of this work. We have used networks of timed automata, extended with integer data variables and arrays of such variables to formally describe real-time systems. To meet requirements from industrial applications, a more sophisticated modelling language will be needed, that supports more data types and constructs for defining abstract data types, such as lists, stacks, queues etc.

In the area of model-checking for timed systems, a main challenge is to develop techniques that combine symbolic representation of timing information with symbolic representation of control-locations. Works in this direction include [WT94, Bal96, BMPY97], however these works are either in an approximation framework or based on symbolically representing the region graph, or a discretization of the region graph.

As a concrete future work, we wish to study the control structure reduction technique presented in paper D further. One possible extension is to identify a minimal

set of covering states that ensures termination and avoids repeated exploration in reachability analysis of timed systems. Recent work for finite-state systems due to Kurshan et.al. [KLM+98] gives some ideas to how a smaller set of covering states can be computed, which are applicable also in a real-time setting. Other promising related techniques for finite-state, such as the use of pseudo-root states due to Parashkevov [PY97], might also be applicable in model-checking algorithms for time-constrained systems.

5 Outline of the Thesis

The rest of this thesis is divided in two main parts: algorithms and data structures, and case studies.

Algorithms and Data Structures

Paper A presents a process algebra of networks of timed automata and a verification algorithm based on on-the-fly and constraint solving techniques for the reachability problem associated with the algebra. As examples, an implementation of the algorithm is applied to verify a version of Fischer's protocol and a railway control system.

In paper B, these ideas are applied to develop a symbolic and on-the-fly model-checking technique for the timed modal logic $\mathcal{L}_s$. The same paper also presents a compositional verification technique for $\mathcal{L}_s$ called quotienting. Both techniques are illustrated by an example.

Paper C describes a diagnostic model-checking technique that generates diagnostic information when a certain property is (or is not) satisfied by a given network of timed automata. It is also shown how the technique was used to debug three early versions of Philips audio-control protocol.

In paper D we present two contributions to the development of memory efficient automatic verification algorithms for timed systems: compact data structure for constraints and the control structure reduction technique. An experiment were the two techniques are applied to six examples from the literature is also presented.

Case Studies

In paper E we study Philips audio-control protocol with bus collision handing. The protocol is modelled with timed automata and analysed in UPPAAL. The same paper also describes the notion of committed locations that is used to accurately model and efficiently verify real-time systems with atomic behaviours.

Paper F presents a rigorous description of the start-up algorithm of a timed division multiple access protocol called DACAPO. The algorithm is analysed in UPPAAL and shown to satisfy its correctness criteria. An upper bound for the start-up algorithm to complete is also derived.

Finally, in paper G we describe an application of UPPAAL where the tool is applied to design and analyse a prototype gearbox controller developed at Mecel AB. It is

also outlined how to check bounded response time properties in a tool like UPPAAL that only provides reachability analysis.

References

[ABL98] Luca Aceto, Augusto Bergueno, and Kim G. Larsen. Model Checking via Reachability Testing for Timed Automata. In Bernard Steffen, editor, *Proc. of the 4*th *Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, number 1384 in Lecture Notes in Computer Science, pages 263–280. Springer–Verlag, 1998.

[Abr95] J.-R. Abrial. Steam-boiler control specification problem. *Int. Seminar on Methods for Semantics and Specification*, June 1995.

[ACD90] Rajeev Alur, Costas Courcoubetis, and David Dill. Model-checking for Real-Time Systems. In *Proc. of Logic in Computer Science*, pages 414–425. IEEE Computer Society Press, June 1990.

[ACHH93] Rajeev Alur, Costas Courcoubetis, Thomas A. Henzinger, and Pei-Hsin Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In R.L̃. Grossman, A. Nerode, A.P̃. Ravn, and H. Rischel, editors, *Hybrid Sysetems*, number 736 in Lecture Notes in Computer Science, pages 209—229. Springer–Verlag, 1993.

[AD90] Rajeev Alur and David Dill. Automata for Modelling Real-Time Systems. In *Proc. of Int. Colloquium on Algorithms, Languages and Programming*, number 443 in Lecture Notes in Computer Science, pages 322–335, July 1990.

[AGU72] A.V. Aho, M.R. Garey, and J.D. Ullman. The Transitive Reduction of a Directed Graph. *SIAM Journal on Computing*, 1(2):131–137, June 1972.

[AH92] Rajeev Alur and Thomas A. Henzinger. Logics and Models of Real-Time: A Survey. In *Proc. of REX Workshop "Real-Time: Theory in Practice"*, number 600 in Lecture Notes in Computer Science. Springer–Verlag, 1992.

[AHV93] Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. Parametric Real-time Reasoning. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on the Theory of Computing*, pages 592–601, 1993.

[AJ98] Parosh Aziz Abdulla and Bengt Jonsson. Verifying Networks of Timed Processes. In Bernard Steffen, editor, *Proc. of the 4*th *Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, number 1384 in Lecture Notes in Computer Science, pages 298–312. Springer–Verlag, 1998.

[AK95] Rajeev Alur and Robert P. Kurshan. Timing Analysis in COSPAN. In Rajeev Alur, Thomas A. Henzinger, and Eduardo D. Sontag, editors, *Proc. of Workshop on Verification and Control of Hybrid Systems III*, number 1066 in Lecture Notes in Computer Science, pages 220–231. Springer–Verlag, October 1995.

[AL92] Martin Abadi and Leslie Lamport. An Old-Fashioned Recipe for Real Time. In *Proc. of REX Workshop "Real-Time: Theory in Practice"*, number 600 in Lecture Notes in Computer Science, 1992.

[AMP97] Eugene Asarin, Oded Maler, and Amir Pnueli. Data–structures for the Verification of timed automata. In *Proc. of the Int. Workshop on Hybrid and Real-Time Systems*, 1997.

[And95] Henrik Reif Andersen. Partial Model Checking. In *Proc. of Symp. on Logic in Computer Science*, 1995.

[Bal96] Felice Balarin. Approximate Reachability Analysis of Timed Automata. In *Proc. of the 17*th *IEEE Real-Time Systems Symposium*, pages 52–61. IEEE Computer Society Press, December 1996.

[Bar94] J.G.P Barnes. *Programming in Ada, Plus and Overview of Ada 9X*. Addison–Wesley, 1994.

[BCM+90] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic Model Checking: 10^{20} states and beyond. In *Proc. of IEEE Symp. on Logic in Computer Science*, 1990.

[BD91] Bernard Berthomieu and Michel Diaz. Modeling and Verification of Time Dependent Systems Using Time Petri Nets. *IEEE Trans. on Software Engineering*, 17(3):259–273, March 1991.

[BDM+98] Marius Bozga, Conrado Daws, Oded Maler, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. Kronos: A Model-Checking Tool for Real-Time Systems. In *Proc. of the 10*th *Int. Conf. on Computer Aided Verification*, number 1427 in Lecture Notes in Computer Science, pages 546–550. Springer–Verlag, 1998.

[Bel57] Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[BFK+98] H. Bowman, G. Faconti, J.-P. Katoen, D. Latella, and M. Massink. Automatic Verification of a Lip Synchronisation Algorithm using Uppaal. In *In Proc. of the 3*rd *Int. Workshop on Formal Methods for Industrial Critical Systems*, 1998.

[BFM98] H. Bowman, G. Faconti, and M. Massink. Specification and Verification of Media Constraints using Uppaal. In *Proc. of the Eurographics Workshop on the Design, Specification and Verification of Interactive Systems*, Eurographics Series. Springer-Verlag, 1998.

[BL96] Johan Bengtsson and Fredrik Larsson. Uppaal a Tool for Automatic Verification of Real-time Systems. Master's thesis, Uppsala University, 1996. Available as `http://www.docs.uu.se/docs/rtmv/bl-report.pdf`.

[BLL+95] Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. Uppaal — a Tool Suite for Automatic Verification of Real–Time Systems. In *Proc. of Workshop on Verification and Control of Hybrid Systems III*, number 1066 in Lecture Notes in Computer Science, pages 232–243. Springer–Verlag, October 1995.

[BLL+98] Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, Wang Yi, and Carsten Weise. New Generation of Uppaal. In *Int. Workshop on Software Tools for Technology Transfer*, June 1998.

[BMPY97] M. Bozga, O. Maler, A. Pnueli, and S. Yovine. Some Progress in the Symbolic Verification of Timed Automata. In Orna Grumberg, editor, *Proc. of the 9*th *Int. Conf. on Computer Aided Verification*, number 1254 in Lecture Notes in Computer Science, pages 179–190. Springer–Verlag, June 1997.

[BPV94] D. Bosscher, I. Polak, and F. Vaandrager. Verification of an Audio-Control Protocol. In *Proc. of Formal Techniques in Real-Time and Fault-Tolerant Systems*, number 863 in Lecture Notes in Computer Science, 1994.

[Bry86] Randal E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Trans. on Computers*, 1986.

[BTY97] Ahmed Bouajjani, Stavros Tripakis, and Sergio Yovine. On-the-Fly Symbolic Model Checking for Real-Time Systems. In *Proc. of the 18*th *IEEE Real-Time Systems Symposium*, pages 25–34. IEEE Computer Society Press, December 1997.

[Büc62] J. Richard Büchi. On a decision method in restricted second order arithmetic. In Ernest Nagel, editor, *Logic, methodology and philosophy of science*, pages 1–11. Stanford University Press, 1962.

[But97] Giorgio C. Buttazzo. *Hard Real-Time Computing Systems: Predictable scheduling algorithms and applications*. Kluwer Academic Publishers, 1997.

[BW90] Allan Burns and Andy Wellings. *Real-time systems and their programming languages*. Addison-Wesley, 1990.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. *Proc. of the 4*th *ACM Symposium on Principles of Programming Languages*, January 1977.

[CCMM97] Antonio Cerone, Alex J. Cowie, George J. Milne, and Philip A. Moseley. Modelling a Time-Dependent Protocol using the Critical Process Algebra. In *Proc. of the Int. Workshop on Hybrid and Real-Time Systems*, number 1201 in Lecture Notes in Computer Science, pages 124–138. Springer–Verlag, March 1997.

[CE81] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using Branching Time Temporal Logic. In *Proc. Workshop on Logics of Programs*, number 131 in Lecture Notes in Computer Science, pages 52–71, Berlin, 1981. Springer–Verlag.

[Cer92] Karlis Cerans. Decidability of Bisimulation Equivalences for Parallel Timer Processes. In *Proc. of CAV'92*, number 663 in Lecture Notes in Computer Science, Berlin, 1992. Springer–Verlag.

[CGL93] Karlis Cerans, Jens Chr. Godskesen, and Kim G. Larsen. Time Modal Specification – Theory and Tools. In *Proc. of the 5*th *Int. Conf. on Computer Aided Verification*, number 697 in Lecture Notes in Computer Science. Springer–Verlag, 1993.

[CHR91] Z. Chaochen, C. A. R. Hoare, and A. P. Ravn. A Calculus of Durations. *Information Processing Letters*, 40(5):269–276, 1991.

[CW96] Edmund M. Clarke and Jeanette M. Wing. Formal Methods: State of the Art and Future Directions. *ACM Computing Surveys*, 28(4):626–643, December 1996.

[Daw98] Conrado Daws. Optikron: A Tool Suite for Enhancing Model-Checking of Real-Time Systems. In *Proc. of the 10*th *Int. Conf. on Computer Aided Verification*, number 1427 in Lecture Notes in Computer Science, pages 542–545. Springer–Verlag, 1998.

[DB96] Pedro R. D'Argenio and Ed Brinksma. A Calculus for Timed Automata. In Bengt Jonsson and Joachim Parrow, editors, *Proc. of Formal Techniques in Real-Time and Fault-Tolerant Systems*, number 1135 in Lecture Notes in Computer Science, pages 110–129. Springer–Verlag, 1996.

[Dij75] Edsger W. Dijkstra. Guarded Commands, Nondeterminiacy and Formal Derivation of Programs. *Communications of the ACM*, 18(8):453–457, 1975.

[Dil89] David Dill. Timing Assumptions and Verification of Finite-State Concurrent Systems. In J. Sifakis, editor, *Proc. of Automatic Verification Methods for Finite State Systems*, number 407 in Lecture Notes in Computer Science, pages 197–212. Springer–Verlag, 1989.

[DKRT96] P.R. D'Argenio, J.-P. Katoen, T.C. Ruys, and J. Tretmans. Modeling and Verifying a Bounded Retransmission Protocol. In *Proc. of COST 247, Int. Workshop on Applied Formal Methods in System Design*, June 1996. Also appears as Technical Report CTIT 96-22, University of Twente, July 1996.

[DKRT97] P.R. D'Argenio, J.-P. Katoen, T.C. Ruys, and J. Tretmans. The bounded retransmission protocol must be on time! In *Proc. of the 3*rd *Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, number 1217 in Lecture Notes in Computer Science, pages 416–431. Springer–Verlag, April 1997.

[DOTY95] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool KRONOS. In Rajeev Alur, Thomas A. Henzinger, and Eduardo D. Sontag, editors, *Proc. of Workshop on Verification and Control of Hybrid Systems III*, number 1066 in Lecture Notes in Computer Science, pages 208–219. Springer–Verlag, October 1995.

[DOY94] C. Daws, A. Olivero, and S. Yovine. Verifying ET-LOTOS programs with KRONOS. In Dieter Hogrefe and Stefan Leue, editors, *Proc. of 7*th *Int. Conf. on Formal Description Techniques*. North–Holland, 1994.

[DS89] Jim Davis and Steve Schneider. An introduction to timed CSP. Technical Report PRG-75, Oxford University Computing Laboratory, 1989.

[DT98] Conrado Daws and Stavros Tripakis. Model checking of real-time reachability properties using abstractions. In Bernard Steffen, editor, *Proc. of the 4*th *Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, number 1384 in Lecture Notes in Computer Science, pages 313–329. Springer–Verlag, 1998.

[DY95] C. Daws and S. Yovine. Two examples of verification of multirate timed automata with KRONOS. In *Proc. of the 16*th *IEEE Real-Time Systems Symposium*, pages 66–75. IEEE Computer Society Press, December 1995.

[DY96] C. Daws and S. Yovine. Reducing the number of clock variables of timed automata. In *Proc. of the 17*th *IEEE Real-Time Systems Symposium*, pages 73–81. IEEE Computer Society Press, 1996.

[GPV94] Aleks Göllü, Anuj Puri, and Pravin Varaiya. Discretization of Timed Automata. In *Proc. of the 33*th *CLC*, 1994.

[Gri94] W.O. David Griffioen. Analysis of an Audio Control Protocol with Bus Collision. Master's thesis, University of Amsterdam, Programming Research Group, 1994.

[Hal93] Nicolas Halbwachs. Delay Analysis in Synchronous Programs. In *Proc. of the 5*th *Int. Conf. on Computer Aided Verification*, number 697 in Lecture Notes in Computer Science, 1993.

[Han91] Hans A. Hansson. *Time and Probability in Formal Design of Distributed Systems*. PhD thesis, Department of Computer Systems, Uppsala University, 1991.

[Hen91] Tomas A. Henzinger. *The Temporal Specification and Verification of Real-Time Systems*. PhD thesis, Stanford University, August 1991.

[HH95] Thomas A. Henzinger and Pei-Hsin Ho. HyTech: The Cornell HYbrid TECHnology Tool. In *Proc. of TACAS, Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, 1995. BRICS report series NS–95–2.

[HHWT95] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. HyTech: The Next Generation. In *Proc. of the 16*th *IEEE Real-Time Systems Symposium*, pages 56–65. IEEE Computer Society Press, December 1995.

[HHWT97] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. HyTech: A Model Checker for Hybird Systems. In Orna Grumberg, editor, *Proc. of the 9*th *Int. Conf. on Computer Aided Verification*, number 1254 in Lecture Notes in Computer Science, pages 460–463. Springer–Verlag, 1997.

[HKV96] Thomas A. Henzinger, Orna Kupferman, and Moshe Vardi. A Space-Efficient On-the-fly Algorithm for Real-Time Model Checking. In *Proc. of CONCUR '96: Concurrency Theory*, number 1119 in Lecture Notes in Computer Science, pages 514–529. Springer–Verlag, August 1996.

[HNSY92] Thomas. A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic Model Checking for Real-Time Systems. In *Proc. of IEEE Symp. on Logic in Computer Science*, 1992.

[Hoa69] C.A.R. Hoare. An axiomatic basis for computer programming. *ACM Communications*, 12(10):576–583, 1969.

[Hoa78] C.A.R. Hoare. Communicating Sequential Processes. *Communications of the ACM*, 21(8):666–677, 1978.

[Hol91] Gerard Holzmann. *The Design and Validation of Computer Protocols*. Prentice Hall, 1991.

[HRP94] N. Halbwachs, P. Raymond, and Y.-E. Proy. Verification of linear hybrid systems by means of convex approximations. In *Static Analysis Symposium*, number 864 in Lecture Notes in Computer Science, pages 223–237, 1994.

[HSLL97] Klaus Havelund, Arne Skou, Kim G. Larsen, and Kristian Lund. Formal Modeling and Analysis of an Audio/Video Protocol: An Industrial Case Study Using Uppaal. In *Proc. of the 18*th *IEEE Real-Time Systems Symposium*. IEEE Computer Society Press, December 1997.

[HWT95] Pei-Hsin Ho and Howard Wong-Toi. Automated Analysis of an Audio Control Protocol. In *Proc. of the 7*th *Int. Conf. on Computer Aided Verification*, number 939 in Lecture Notes in Computer Science. Springer–Verlag, 1995.

[HWT96] Thomas A. Henzinger and Howard Wong-Toi. Using HyTech to Synthesize Control Parameters for a Steam Boiler Control. In *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler*, number

1165 in Lecture Notes in Computer Science, pages 265–282. Springer–Verlag, 1996.

[JBW+96] Mathai Joseph, Alan Burns, Andy Wellings, Krithi Ramamritham, Jozef Hooman, Steve Schneider, Zhiming Liu, and Henk Schepers. *Real-time Systems Specification, Verification and Analysis.* Prentice Hall, 1996.

[JLS96] Henrik E. Jensen, Kim G. Larsen, and Arne Skou. Modelling and Analysis of a Collision Avoidance Protocol Using SPIN and UPPAAL. In *Proc. of 2*nd *Int. Workshop on the SPIN Verification System*, pages 1–20, August 1996.

[Jon87] Bengt Jonsson. *Compositional Verification of Distributed Systems.* PhD thesis, Department of Computer Systems, Uppsala University, 1987.

[KL94] Inhye Kang and Insup Lee. An Efficient Generation of the Timed Reachability Graph for the Analysis of Real-Time Systems. Technical Report MS-CIS-94-36, Dept. of Comp. and Infor. Science, Univ. of Penn, 1994.

[KL96] Inhye Kang and Insup Lee. An Efficient State Space Generation for Analysis of Real-Time Systems. In *International Symposium on Software Testing and Analysis*, January 1996. TREAT is available at `http://www.cis.upenn.edu/~lee/inhye/treat.html`.

[KLL+97] Kåre J. Kristoffersen, Francois Laroussinie, Kim G. Larsen, Paul Pettersson, and Wang Yi. A Compositional Proof of a Real-Time Mutual Exclusion Protocol. In *Proc. of the 7*th *Int. Joint Conf. on the Theory and Practice of Software Development*, April 1997.

[KLM+98] R. Kurshan, V. Levin, M. Minea, D. Peled, and H. Yenigün. Static partial order reduction. In *Proc. of the 4*th *Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, number 1384 in Lecture Notes in Computer Science, pages 281–297. Springer–Verlag, March 1998.

[KP95] Kåre Jelling Kristoffersen and Paul Pettersson. Modelling and Analysis of a Steam Generator using UPPAAL. In *Proc. of the 7*th *Nordic Workshop on Programming Theory*, pages 156–164, November 1995. Report 86, Chalmers University of Technology.

[Lam77] Leslie Lamport. Proving the correctness of multiprocess programs. *IEEE Trans. on Software Engineering*, 3(2):125–143, March 1977.

[Lam80] Leslie Lamport. 'Sometimes' is sometimes 'not never'. In *Proc. 7*th *ACM Symp. on Principles of Programming Languages*, pages 174–185, 1980.

[Lar86] Kim G. Larsen. *Context-Dependent Bisimulation Between Processes.* PhD thesis, University of Edinburgh, Mayfield Road, Edinburgh, Scotland, 1986.

[LL95] Francois Laroussinie and Kim G. Larsen. Compositional Model Checking of Real Time Systems. In *Proc. of CONCUR '95: Concurrency Theory*, Lecture Notes in Computer Science. Springer–Verlag, 1995.

[LL98] Francois Laroussinie and Kim G. Larsen. CMC: a tool for compositional model-checking of real-time systems. In *Proc. of Joint International Conference on Formal Description Techniques and Protocol Specification, Testing, and Verification*, 1998.

[LPY97a] Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a Nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1(1–2):134–152, October 1997.

[LPY97b] Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL: Status and Developments. In Orna Grumberg, editor, *Proc. of the 9*th *Int. Conf. on Computer Aided Verification*, number 1254 in Lecture Notes in Computer Science, pages 456–459. Springer–Verlag, June 1997.

[LSW97] Kim G. Larsen, Bernard Steffen, and Carsten Weise. Continuous modeling of real-time and hybrid systems: from concepts to tools. *Int. Journal on Software Tools for Technology Transfer*, 1(1–2):64–85, December 1997.

[LY93] Kim G. Larsen and Wang Yi. Time Abstracted Bisimulation: Implicit Specification and Decidability. In *MFPS93*, number 802 in Lecture Notes in Computer Science. Springer–Verlag, 1993.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice Hall, Englewood Cliffs, 1989.

[MY96] Oded Maler and Sergio Yovine. Hardware timing verification using KRONOS. In *Proc. of the IEEE 7*th *Israeli Conference on Computer Systems and Software Engineering*. IEEE Computer Society Press, 1996.

[Nan92] Nancy Lynch and Frits Vaadrager. Forward and backward simulations for timing-based systems. In J.W. de Bakker, C. Huizing, W.-P de Roever, and G. Rozenberg, editors, *Proc. of workshop on Stepwise Refinement of Distributed Systems*, number 600 in Lecture Notes in Computer Science, pages 397–446. Springer–Verlag, 1992.

[Nil93] Martin Nilsson. Piecewise Linear Constraints and Entailment. Technical report, Swedish Institute of Computer Science, August 1993.

[NRJV90] X. Nicollin, J. L. Richierand, J.Sifakis, and J. Voiron. ATP: an algebra for timed processes. *Proc. of the IFIP TC 2 Working Conf. on Programming Concepts and Methods*, 1990.

[NSY92a] X. Nicollin, J. Sifakis, and S. Yovine. From ATP to Timed Graphs and Hybrid Systems. In *Proc. of the 1991 REX worksop "Real-Time: Theory in Practice"*, volume 600 of *Lecture Notes in Computer Science*, pages 549–572. Springer–Verlag, 1992.

[NSY92b] Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Compiling Real-Time Specifications into Extended Automata. *IEEE Transactions on Software Engineering. Special Issue on Real-Time Systems*, 18(9):794–804, September 1992.

[OG76] S. Owicki and D. Gries. An Axiomatic Proof Technique for Parallel Programs I. *Acta Informatica*, 6(4):319–340, 1976.

[Pnu77] Amir Pnueli. The Temporal Logic of Programs. In *Proc. 16*th *Annual Symp. Foundations of Computer Science*, pages 46–57, 1977.

[Pnu86] Amir Pnueli. Applications of temporal logic to the specification and verification of reactive systems: a survey of current trends. In J.W. de Bakker, W.P de Roever, and G. Rozenberg, editors, *Current Trends in Concurrency: Overviews and Tutorials*, number 224 in Lecture Notes in Computer Science, pages 510–584. Springer–Verlag, 1986.

[PY97] A.N. Parashkevov and J. Yantchev. Space Efficient Reachability Analysis Through Use of Pseudo–Root States. In *Proc. of the 3*rd *Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, number 1217 in Lecture Notes in Computer Science, 1997.

[QS82] J. P. Queille and J. Sifakis. Specification and verification of concurrent programs in CESAR. In *Proc. 5*th *Int. Symp. on Programming*, number 137 in Lecture Notes in Computer Science, pages 195–220, Berlin, 1982. Springer–Verlag.

[RLST95] B. Rostamzadeh, H. Lönn, R. Snedsböl, and J. Torin. DACAPO: A Distributed Computer Archiecture for Safety-Critical Control Applications. In *Proc. of IEEE Int. Symp. on Intelligent Vehicles*, 1995.

[RM94] Tomas G. Rokicki and Chris J. Myers. Automatic verification of timed circuits. In David L. Dill, editor, *Proc. of the 6*th *Int. Conf. on Computer Aided Verification*, number 818 in Lecture Notes in Computer Science, pages 468–480. Springer–Verlag, 1994.

[Rok93] Tomas Gerhard Rokicki. *Representing and Modeling Digital Circuits.* PhD thesis, Stanford University, 1993.

[Sif77] Joseph Sifakis. Use of Petri Nets for performance evaluation. In H. Beilner and E. Gelenbe, editors, *Measuring, Modelling and Evaluating Computer Systems*, pages 75–93. North-Holland, 1977.

[SMF97] Thomas Stauner, Olaf Müller, and Max Fuchs. Using HyTech to Verify an Automotive Control System. In *Proc. Hybrid and Real-Time Systems, Grenoble, March 26-28, 1997.* Technische Universität München, Lecture Notes in Computer Science, Springer, 1997.

[SS95] Oleg V. Sokolsky and Scott A. Smolka. Local Model Checking for Real-Time Systems. In *Proc. of the 7*th *Int. Conf. on Computer Aided Verification*, number 939 in Lecture Notes in Computer Science, pages 211–224. Springer–Verlag, 1995.

[Sti86] C. Stirling. A Compositional Reformulation of Owicki–Gries's Partial Correctness Logic for a Concurrent While Language. In *Proc. of Int. Colloquium on Algorithms, Languages and Programming*, number 226 in Lecture Notes in Computer Science. Springer–Verlag, 1986.

[Sto96] Neil Storey. *Safety-Critical Computer Systems.* Addison–Wesley, 1996. ISBN 0-201-42787-7.

[SV96] Jan Springintveld and Frits Vaandrager. Minimizable Timed Automata. In Bengt Jonsson and Joachim Parrow, editors, *Proc. of Formal Techniques in Real-Time and Fault-Tolerant Systems*, number 1135 in Lecture Notes in Computer Science, pages 130–147. Springer–Verlag, 1996.

[TC96] Stavros Tripakis and Costas Courcoubetis. Extending Promela and Spin for Real-Time. In *Proc. of the 2*nd *Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, pages 329–348. Springer–Verlag, March 1996.

[TH96] Ken Tindell and Hans Hansson. Real time systems by fixed priority scheduling. Draft, 1996.

[TY98] Stavros Tripakis and Sergio Yovine. Verification of the Fast Reservation Protocol with Delayed Transmission using the tool Kronos. In *Proc. of the 4*th *IEEE Real-Time Technology and Applications Symposium.* IEEE Computer Society Press, June 1998.

[VW86] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. of Symp. on Logic in Computer Science*, pages 322–331, June 1986.

[WH98] Farn Wang and Pao-Ann Hsiung. Automatic Verification on the Large. In *Proc. of the 3*rd *IEEE High-Assurance Systems Engineering Symposium*, November 1998.

[WT94] Howard Wong-Toi. *Symbolic Approximations for Verifying Real-Time Systems.* PhD thesis, Stanford University, November 1994.

[Yi91] Wang Yi. *A Calculus of Real Time Systems.* PhD thesis, Department of Computer Science, Chalmers University of Technology, 1991.

[YT96] Sergio Yovine and Stavros Tripakis. Analysis of timed systems based on time-abstracting bisimulations. In Rajeev Alur and Thomas A. Henzinger, editors, *Proc. of the 8*th *Int. Conf. on Computer Aided Verification*, number 1102 in Lecture Notes in Computer Science, pages 232–243. Springer–Verlag, 1996.

Part II
Algorithms and Data Structures

Paper A

Wang Yi, Paul Pettersson, and Mats Daniels. Automatic Verification of Real-Time Communicating Systems by Constraint-Solving. In Dieter Hogrefe and Stefan Leue, editors, *Proceedings of the 7*th *International Conference on Formal Description Techniques*, pages 223–238. North-Holland 1994.

Automatic Verification of Real-Time Communicating Systems by Constraint-Solving

Wang Yi, Paul Pettersson and Mats Daniels

Department of Computer Systems, Uppsala University. Sweden.
Email: {yi,paupet,matsd}@docs.uu.se.

Abstract. In this paper, an algebra of timed processes with real-valued clocks is presented, which may serve as a description language for networks of timed automata. We show that requirements such as "a process will never reach an undesired state" can be verified by solving a simple class of constraints on the clock-variables. A symbolic on-the-fly reachability algorithm for the language has been developed and implemented as a software tool based on constraint-solving techniques. To our knowledge, this is the first on-the-fly verification algorithm for timed automata. In fact, the tool is the very first implementation of the UPPAAL tool.

As examples, we model and verify safety properties of a real-time mutual exclusion protocol and a railway crossing controller.

1 Introduction

During the past few years, various formal techniques for modeling and verifying real-time systems have been put forwards, e.g. automata based [ACD90, ACH+92, AD90, HNSY92] and process algebra based [Cer92, CGL93, HLY92, MT91, Yi91]. One of the most successful approaches is the *timed automata* model due to Alur and Dill [ACD90], which is the classical finite-state automaton model extended with clock variables modeling time delays.

In this paper, we study real-time communicating systems. Such a system consists of a number of components with their own or shared clocks. The components may communicate with each other and the environment through channels according to the timing constraints on the values of the clocks. Naturally, we can use timed automata to describe the components. However, it is not obvious how to combine the component descriptions in a description of the whole system. Traditionally, the parallel composition of timed automata is interpreted as logical conjunction, which is similar to the strong (multi-) synchronisation operator from process algebras, defined by the rule:

$$\frac{P \stackrel{a}{\Leftrightarrow\!\!\rightarrow} P' \qquad Q \stackrel{a}{\Leftrightarrow\!\!\rightarrow} Q'}{P \;\&\; Q \stackrel{a}{\Leftrightarrow\!\!\rightarrow} P' \;\&\; Q'}$$

Intuitively, it means that the whole system described by $P \;\&\; Q$ may make a move (i.e. do an a) only if the components described by P and Q can do the same. That is, all components of a concurrent system must synchronise on every action at every time point. This seems to be a strong restriction for practical applications of timed automata. Real systems are often highly distributed and in many cases, a system component may only want to communicate with the environment or a particular

component, without synchronising with the others. Therefore, we introduce a CCS-like parallel composition operator [Mil89] for timed automata, to describe one-to-one communication and interleaving.

As the first contribution of this paper, we present an algebra for networks of timed automata, which provides a number of algebraic operators including the parallel composition operator to model communication and concurrency. The operators can be used to construct complex automata (i.e. complex system descriptions) in terms of simpler ones (i.e. component descriptions). Thus, the algebra may serve as a *structural* description language for real-time communicating systems.

As the second contribution, we develop an on-the-fly verification algorithm based on constraint-solving techniques, for the type of systems described above. There have been a number of verification techniques developed for timed automata, e.g. [ACH+92, ACD90, HNSY92]. However, these algorithms always construct the product of the automata before checking properties of the system. Even for moderately sized systems, explicit representation and exploration of the product automaton is infeasible as it grows exponentially with both the number of components and the number of clocks in the system. Though there have been efficient minimisation techniques, such as [ACH+92], the problem of state-space explosion is still an obstacle for automatic verification.

It has been a well recognised fact (e.g. [Hal93]) that the practical goal of verification of real-time systems is often to verify simple logical properties, which do not need the whole power of model-checking (e.g. for timed CTL). We shall only consider simple safety-properties, which can be verified by checking if a certain set of states of the system is reachable or not. For instance, a railway control system (see Section 6) should guarantee that "at most one train can cross a critical point at the same time". This is a typical safety-property meaning that bad things can never happen. However, we can also verify properties requiring that a good thing will eventually happen within a certain time limit. For example, "a train should be able to pass a critical point (such as a bridge), within a bounded delay".

We shall present a symbolic on-the-fly verification algorithm for networks of timed automata. It is *symbolic* in the sense that the inherently infinite state-space of timed automata is finitely partitioned into subsets which are represented and manipulated using a class of linear constraints, known as *difference bound matrices* [Bel57, Dil89]. This allows the partitioning to take into account both the automata and the logical property currently being checked, to make the partitioning as coarse (and small) as possible. The algorithms operates in an *on-the-fly* manner [Hol91] in the sense that the checking of a particular property and the construction of the reachable part of the (symbolic) state-space is performed simultaneously. This avoids generating unnecessary parts of the state-space as the algorithm is stopped when the truthhood of the currently checked property is determined, in contrast to the traditional approach of constructing the full state-space of the system before the checking.

The rest of the paper is organised as follows: In section 2, we present an algebra of timed processes, in which a syntactical term describes a network of timed automata; any timed automaton can be expressed in the algebra. Section 3 and 4 describe two symbolic semantics of processes, of which the latter is shown to yield a finite number

of symbolic states. In section 5, we study the reachability problem associated with the algebra. An algorithm based on the finite symbolic semantics is presented, and proved to be sound and complete. In section 6, as examples, we study a simple railway control system and variant of Fischer's mutual exclusion protocol. Section 7 concludes the paper.

2 An Algebra of Processes with Clocks

Process algebras provide a clean and general paradigm for compositional specification of communicating processes. In this section we present an algebra of timed automata, serving as a structural specification language for real-time communicating systems. The idea is to use algebraic operators to construct complex system specifications in terms of simpler ones (or component specification).

2.1 Syntax

Traditionally, a prefix expression $\alpha.P$ in process algebras describes a process which may perform an α-transition and then continue with P. But no timing information is given on when the transition may be taken.

Following Alur and Dill [AD90], we assume a set of clocks to specify timing constraints on transitions. Conceptually, the clocks may be considered as the system clocks of a concurrent system, owned or shared by processes in the system. The processes may test the clocks by comparing the clock values with integer constants and reset the clocks (i.e. assigning clock values to 0). Further, assume that all clocks proceed at the same rate and measure the amount of time that has been elapsed since they were reset or started.

We extend the action prefix $\alpha.P$ to the form $(g, \alpha, r).P$ where g is a predicate over the clock values and r is a subset of clocks to be reset. Intuitively, $(g, \alpha, r).P$ describes a timed process which may perform an α-transition instantaneously when g is true of the current clock values and then continue as P with the clocks in r being reset (and the other clocks proceeding with their old values).

We also introduce a way to force processes to make progress. We use an invariant operator in the form $g \rhd\!\gg P$, where g is a predicate over clock values [HNSY92, DB96]. Intuitively, the process may idle or go on as P while g is satisfied by the system clocks.

Definition 2.1 (Clock Constraints) *Let $\mathcal{C}$ denote a set of clocks, ranged over by x, y ,z. We use $\mathcal{B}(\mathcal{C})$ to stand for the set of logical formulae generated by the following syntax:*

$$g ::= x \sim n \mid x \Leftrightarrow y \sim n \mid g \wedge g$$

where $\sim \in \{<, >, =, \leq, \geq\}$ and n is a natural number. □

We shall use $\mathtt{tt}$ to denote a clock constraint like $x \geq 0$ that is always satisfied, and $\mathtt{ff}$ for $x < 0$ that is always false as clocks can only have non-negative values. Note that we could allow a more general form of formulas such as disjunction $g \vee g$. However, it will

not give more expressive power to the description language we are going to develop. In fact, logical disjunction can be modeled by the behavioural choice operator.

The language is essentially CCS [Mil89] extended with the timed action prefix $(g, \alpha, r).P$ and the invariant operator $g \rhd\!\gg P$. Let $\mathcal{A}$ be a finite set of actions ranged over by α, β etc. We use $\mathcal{L} = \{\alpha \mid \alpha \in \mathcal{A}\} \cup \{\bar{\alpha} \mid \alpha \in \mathcal{A}\}$[1] with $\bar{\bar{\alpha}} = \alpha$ for representing external actions, a distinct symbol τ representing internal actions, and $\mathcal{A}ct = \mathcal{L} \cup \{\tau\}$ ranged over by a, b for representing both internal and external actions. Further, assume a set of process variables ranged over by X, Y (and sequences of letters).

We shall see that the algebraic structure of a process expression P represents the control-structure of a process. This will be clear when we present the operational semantics. We adopt a two-phase syntax according to two types of control-structures: *regular* and *concurrent*.

We start with processes whose control-structure are regular in the sense that no concurrency is involved. The regular process expressions are generated by the following grammar:

$$E \ ::= \ \mathsf{nil} \mid (g, a, r).E \mid g \rhd\!\gg E \mid E + F \mid X \mid X \stackrel{def}{=} E$$

We shall restrict expressions to be *well-guarded* in the following sense:

Definition 2.2 (Well-Guarded Expressions) *X is well-guarded in E if and only if every free occurrence of X in E is within a subexpression (a guard) of the form $(g, a, r).F$. E is well-guarded if and only if every free variable in E is well-guarded in E, and for every subexpression of the form $X \stackrel{def}{=} F$ in E, X is well-guarded in F.* □

Let $\mathcal{P}$ denote the set of well-guarded expressions generated by the grammar above. We call $\mathcal{P}$ *regular timed processes*.

We shall study concurrent processes in the form $(P_1 \mid ... \mid P_n)\backslash L$, where $P_i \in \mathcal{P}$ describing the components and $L \subseteq \mathcal{L}$ representing the set of internal channels connecting the components. We use $\boldsymbol{\mathcal{P}}$ to denote the set of timed concurrent processes, ranged over by P, Q and R.

For simplicity, we have ignored the relabelling operator. The results of this paper can easily be extended to more general types of processes modeled by the combination of parallel composition, restriction and relabelling.

2.2 Operational Semantics

We interpret $\boldsymbol{\mathcal{P}}$ using clock assignments. Let $\mathbb{R}_+$ stand for the non-negative real numbers. A *clock assignment* $u : \mathcal{C} \mapsto \mathbb{R}_+$ is a function mapping each clock x to a non-negative real $u(x)$. Assume that $d \in \mathbb{R}_+$ and $r \subseteq \mathcal{C}$ is a set of clocks. We use $u \oplus d$ to denote the clock assignment which maps each clock x to $u(x) + d$, and $r[u]$ to denote the clock assignment which maps x to 0 if $x \in r$ and to $u(x)$ otherwise.

[1] The action $\bar{\alpha}$ is called the co-action of α. In our examples, we shall use $\alpha!$ instead of $\bar{\alpha}$ to denote an *output* event and $\alpha?$ instead of α to denote an *input* event.

Furthermore, given a clock constraint $g \in \mathcal{B}(\mathcal{C})$, we write $g(u)$ to mean the truth value of g, relative to assignment u.

To interpret $\mathcal{P}$ we also define for each control-node $E \in \mathcal{P}$ an *invariant condition*, denoted $I(E)$. Intuitively, $I(E)$ restricts the amount of time the process may idle in E, i.e. it must switch to another control-node while $I(E)$ still holds. It is defined as follows:

Definition 2.3 (Invariant Condition) *Assume $E, F \in \mathcal{P}$. Let $I(E)$ be the invariant condition of E, defined inductively as:*

$$\begin{aligned} I(\mathsf{nil}) &= \mathsf{tt} \\ I((g,a,r).E) &= \mathsf{tt} \\ I(g \rhd\!\gg E) &= g \wedge I(E) \\ I(E+F) &= I(E) \wedge I(F) \\ I(X) &= I(E) \text{ if } X \stackrel{def}{=} E \\ I(E|F) &= I(E) \wedge I(F) \\ I(E\backslash L) &= I(E) \end{aligned}$$

□

A *state* of a process is a pair (P,u) where $P \in \mathcal{P}$ stands for the current control-node and u denotes the current clock values. A process may make two types of transitions from state to state:

Definition 2.4 (Transition Rules) *Assume $a \in \mathcal{A}ct$ and $d \in \mathbb{R}_+$:*

- *(Action) $(P,u) \stackrel{a}{\rightsquigarrow} (P',u')$ following the rules defined in Table 1.*
- *(Delay) $(P,u) \stackrel{d}{\rightsquigarrow} (P, u \oplus d)$ if $I(E)(u)$ and $I(E)(u \oplus d)$.*

We will use $(P^0,u^0) \rightsquigarrow^n (P^n,u^n)$ to denote $(P^0,u^0) \stackrel{\sigma_1}{\rightsquigarrow} (P^1,u^1) \stackrel{\sigma_2}{\rightsquigarrow} \ldots \stackrel{\sigma_{n-1}}{\rightsquigarrow} (P^{n-1}, u^{n-1}) \stackrel{\sigma_n}{\rightsquigarrow} (P^n,u^n)$ for $\sigma_i \in \mathcal{A}ct \cup \mathbb{R}_+$, and $(P^0,u^0) \rightsquigarrow^\star (P^f,u^f)$ if $(P^0,u^0) \rightsquigarrow^n (P^f,u^f)$ for some finite number n. □

The delay transition relation describes the pure passing of time; the action transition relation describes the instantaneous occurrence of actions and, possibly, the resetting of clocks.

Example 2.1 *Consider the algebraic terms X, P and Q:*

$$\begin{aligned} X &\stackrel{def}{=} (y \leq 1) \rhd\!\gg (\mathsf{tt}, \tau, \{\}).P \\ P &\stackrel{def}{=} (x \leq 2) \rhd\!\gg \Big((x < 1, \tau, \{\}).Q + (x \geq 1, a, \{x\}).P \Big) \\ Q &\stackrel{def}{=} \mathsf{nil} \end{aligned}$$

The control-node X will become P by doing a τ-action before the clock value of y becomes greater than 1. If the value of y is greater than 1, X is deadlocked. However, when the value of x is greater than, or equal to 1, the control-node of P will remain the same, i.e. P, but the τ-action will be disabled and the a-action will be enabled

$$\frac{g(u)}{((g,\alpha,r).E,u) \overset{a}{\rightsquigarrow} (E,r[u])} \qquad \frac{(E,u) \overset{a}{\rightsquigarrow} (E,u')}{(g \rhd\!\!\gg E,u) \overset{a}{\rightsquigarrow} (E,u')}$$

$$\frac{(F,u) \overset{a}{\rightsquigarrow} (F',r[u])}{(E+F,u) \overset{a}{\rightsquigarrow} (F',r[u])} \qquad \frac{(E,u) \overset{a}{\rightsquigarrow} (E',r[u])}{(E+F,u) \overset{a}{\rightsquigarrow} (E',r[u])}$$

$$\frac{(E,u) \overset{a}{\rightsquigarrow} (E',r[u])}{(X,u) \overset{a}{\rightsquigarrow} (E',r[u])} \quad [X \overset{def}{=} E]$$

$$\frac{(E,u) \overset{a}{\rightsquigarrow} (E',r[u])}{(E|F,u) \overset{a}{\rightsquigarrow} (E'|F,r[u])} \qquad \frac{(F,u) \overset{a}{\rightsquigarrow} (F',r[u])}{(E|F,u) \overset{a}{\rightsquigarrow} (E|F',r[u])}$$

$$\frac{(E,u) \overset{a}{\rightsquigarrow} (E',r[u]) \quad (F,u) \overset{\bar{a}}{\rightsquigarrow} (F',q[u])}{(E|F,u) \overset{\tau}{\rightsquigarrow} (E'|F',(r \cup q)[u])}$$

$$\frac{(E,u) \overset{a}{\rightsquigarrow} (E',r[u])}{(E\backslash L,u) \overset{a}{\rightsquigarrow} (E'\backslash L,r[u])} \quad [a,\bar{a} \notin L]$$

Table 1: The Action Transition Relation.

while x is less than, or equal to 2. For instance, X can perform the following sequence of transitions:

$$(X,0,0) \overset{\tau}{\rightsquigarrow} (P,0,0) \overset{m}{\rightsquigarrow} (P,m,m) \overset{a}{\rightsquigarrow} (P,0,m) \overset{\tau}{\rightsquigarrow} (Q,0,m)$$

for all $m \in [1,2]$.

3 Symbolic Semantics of Processes

Clearly, the concrete semantics of processes defined in the previous section yields an infinite transition system due to the real-valued clocks. In this section we shall give a symbolic semantics of processes which will be used in the next section to develop a finite partitioning of the state-space.

We consider symbolic states that are sets of concrete states sharing the same control-node. To represent sets of clock assignments we use clock constraints $\mathcal{B}(\mathcal{C})$ introduced in Section 2.1. Let D range over $\mathcal{B}(\mathcal{C})$. The key idea is to let D represent the set of clock assignments that are its solutions, and use states in the form (P,D) to *symbolically* represent the set of all states (P,u) such that u satisfies D.

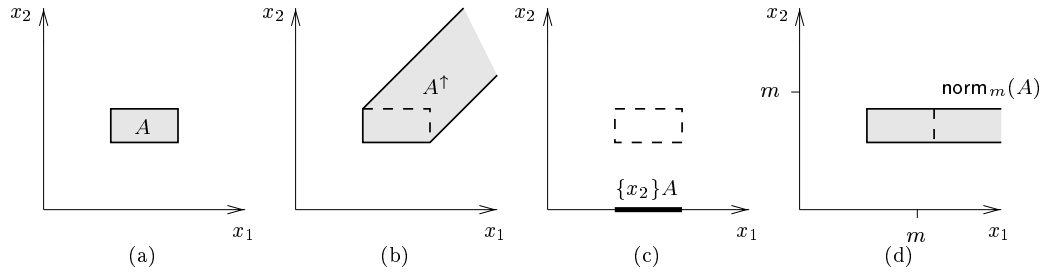


Figure 1: Operations on Clock Constraints.

3.1 Operations on Clock Constraints

We will need a few *operations* on clock constraints to define the symbolic semantics. Given a clock constraint D we call the set of clock assignments satisfying D, the *solution set* of D.

Definition 3.1 *Let A and A' be solution sets of clock constraint D and D'. We define:*

$$\begin{array}{rcl} A \wedge A' & = & \{\ u \mid u \in A \text{ and } u \in A'\ \} \\ A^{\uparrow} & = & \{\ u \oplus d \mid d \in \mathbb{R}_+ \text{ and } u \in A\} \\ \{x\}A & = & \{\ x[u] \mid u \in A\ \} \end{array}$$

□

First note that $A \wedge A'$ is simply the intersection of the two sets. Consider the set A for the case of two clocks, shown in Figure 1(a). The two operations $A^{\uparrow}$ and $\{x\}A$ are illustrated in (b) and (c) of Figure 1 respectively[2]. Intuitively, $A^{\uparrow}$ is the largest set of clock assignments that will eventually be reached from A after some delay, and the reset-operation $\{x_2\}A$ is the projection of A down on the x_1-axis. We extend the reset-operation $\{x\}A$ to sets of variables. Assume that r is a set of clock variables and $r = \{x\} \cup r'$. We define $r(A) = \{x\}(\{r'\}A)$ and $\{\}A = A$. In order to save notation, from now on we shall simply use $D \wedge D'$, $D^{\uparrow}$ and $\{x\}D$ to denote the solution sets $A \wedge A$, $A^{\uparrow}$ and $\{x\}A$.

We also need two *predicates* over clock constraints in the semantics. We write $D \subseteq D'$ to mean that the solution set of D is included in the solution set of D' and $D = \emptyset$ to mean that the solution set of D is empty (i.e. D is not satisfiable).

3.2 Symbolic Transition Rules

In this section we will define the transition rules for symbolic states. First, we need to study the control-structure of processes more carefully.

We will interpret algebraic terms $\mathcal{P}$ as timed automata with location invariants[3] [AD90, HNSY92]. It should be obvious that each term $P \in \mathcal{P}$ describes a timed automaton with location invariants, i.e. $\langle \mathcal{P}, P, \Leftrightarrow\!\!\rightarrow, I \rangle$, where $\mathcal{P}$ (the set of

[2]Figure 1(d) is to illustrate the normalisation operator that will be introduced in the next section.

[3]Timed automata with location invariants are called "timed safety automata" in [HNSY92].

$$\frac{}{(g,a,r).E \overset{g\,a\,r}{\Leftrightarrow\!\!\rightarrow} E} \qquad \frac{E \overset{g\,a\,r}{\Leftrightarrow\!\!\rightarrow} E'}{\psi \rhd\!\!> E \overset{g\,a\,r}{\Leftrightarrow\!\!\rightarrow} E'}$$

$$\frac{E \overset{g\,a\,r}{\Leftrightarrow\!\!\rightarrow} E'}{E+F \overset{g\,a\,r}{\Leftrightarrow\!\!\rightarrow} E'} \qquad \frac{F \overset{g\,a\,r}{\Leftrightarrow\!\!\rightarrow} F'}{E+F \overset{g\,a\,r}{\Leftrightarrow\!\!\rightarrow} F'}$$

$$\frac{E \overset{g\,a\,r}{\Leftrightarrow\!\!\rightarrow} E'}{X \overset{g\,a\,r}{\Leftrightarrow\!\!\rightarrow} E'} \quad [\ X \overset{def}{=} E\ ]$$

$$\frac{E \overset{g\,a\,r}{\Leftrightarrow\!\!\rightarrow} E'}{E|F \overset{g\,a\,r}{\Leftrightarrow\!\!\rightarrow} E'|F} \qquad \frac{F \overset{g\,a\,r}{\Leftrightarrow\!\!\rightarrow} F'}{E|F \overset{g\,a\,r}{\Leftrightarrow\!\!\rightarrow} E|F'}$$

$$\frac{E \overset{g\,a\,r}{\Leftrightarrow\!\!\rightarrow} E' \quad F \overset{f\,\bar{a}\,s}{\Leftrightarrow\!\!\rightarrow} F'}{E|F \overset{h\,\tau\,q}{\Leftrightarrow\!\!\rightarrow} E'|F'} \left[\begin{array}{c} h = g \wedge f \\ q = r \cup s \end{array}\right]$$

$$\frac{E \overset{g\,a\,r}{\Leftrightarrow\!\!\rightarrow} E'}{E\backslash L \overset{g\,a\,r}{\Leftrightarrow\!\!\rightarrow} E'\backslash L} \quad [\ a, \bar{a} \notin L\ ]$$

Table 2: Transition Rules for Control-Nodes.

algebraic terms) is the set of nodes, $P \in \boldsymbol{\mathcal{P}}$ the initial node, $\Leftrightarrow\!\!\rightarrow$ is the least transition relation defined by the rules in Table 2, and $I : \boldsymbol{\mathcal{P}} \mapsto \mathcal{B}(\mathcal{C})$ is the invariant assignments function as defined in Definition 2.3. In particular, note that $\boldsymbol{\mathcal{P}}$ is finite. Note also that the terms $P \in \boldsymbol{\mathcal{P}}$ are in the form $P = (P_1|\ldots|P_n)\setminus L$, where $P_i \in \mathcal{P}$ and $L \subseteq \mathcal{L}$. Thus, rather than describing a single timed automaton, an algebraic term e.g. P may describe a whole *network of timed automata* P_i that communicate by synchronising pairwise on the internal channels (in L) connecting them.

Example 3.1 *Reconsider the algebraic terms X, P and Q of Example 2.1. The timed automaton A_X described by X is shown in Figure 2. It has three control-nodes, X, P and Q, two clocks x and y, and three edges. The edge between P and Q has $x < 1$ as guard, τ as action and empty reset set $\{\}$. The invariant condition of the control-node P is $x \leq 2$.*

We are now ready to define the symbolic semantics of processes. It is given by a transition system with the set of states being the *symbolic states* (P, D) and the transition relation defined by the following two rules:

Definition 3.2 (Symbolic Transition Rules) *Assume $a \in \mathcal{A}ct$ and a new symbol ε representing delays.*

- *(Action)* $(P, D) \overset{a}{\Leftrightarrow\!\!\rightarrow} (P', r(D \wedge g))$ *if* $P \overset{g\,a\,r}{\Leftrightarrow\!\!\rightarrow} P'$.

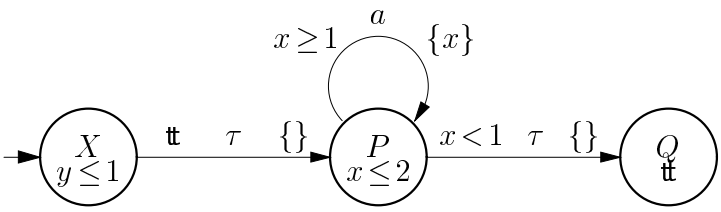


Figure 2: Timed Automaton A_X.

- *(Delay)* $(P, D) \stackrel{\varepsilon}{\Leftrightarrow\!\!\rightarrow} (P, D^{\uparrow P})$,

where $D^{\uparrow P} = (D \wedge I(P))^{\uparrow} \wedge I(P)$, *and* $I(P)$ *is the invariant condition of* P.

We will use $(P^0, D^0) \Leftrightarrow\!\!\rightarrow^n (P^n, D^n)$ *to denote an alternating sequence in the form* $(P^0, D^0) \stackrel{\varepsilon}{\Leftrightarrow\!\!\rightarrow} (P^1, D^1) \stackrel{a_2}{\Leftrightarrow\!\!\rightarrow} (P^2, D^2) \stackrel{\varepsilon}{\Leftrightarrow\!\!\rightarrow} \ldots \stackrel{a_{n-1}}{\Leftrightarrow\!\!\rightarrow} (P^{n-1}, D^{n-1}) \stackrel{\varepsilon}{\Leftrightarrow\!\!\rightarrow} (P^n, D^n)$ *for* $a_i \in \mathcal{A}ct$, *and* $(P^0, D^0) \Leftrightarrow\!\!\rightarrow^\star (P^f, D^f)$ *if* $(P^0, D^0) \Leftrightarrow\!\!\rightarrow^n (P^n, D^n)$ *for some* n. $\Box$

Intuitively, in the action transition relation $(P', r(D \wedge g))$ is the strongest post-condition of (P, D) after a transition $P \stackrel{g\,a\,r}{\Leftrightarrow\!\!\rightarrow} P'$. In the delay transition relation, $D^{\uparrow P}$ is the largest set of clock assignments that can be reached from D by delaying in P while the invariant condition $I(P)$ still holds.

Example 3.2 *Consider the timed automaton of Figure 2. It has the following typical symbolic transition sequence:*

$$
\begin{array}{l}
(X, (x = y = 0)) \stackrel{\varepsilon}{\Leftrightarrow\!\!\rightarrow} (X, (x = y \wedge y \leq 1)) \stackrel{\tau}{\Leftrightarrow\!\!\rightarrow} \\
(P, (x = y \wedge y \leq 1)) \stackrel{\varepsilon}{\Leftrightarrow\!\!\rightarrow} (P, (x = y \wedge y \leq 2)) \stackrel{a}{\Leftrightarrow\!\!\rightarrow} \\
(P, (x = 0 \wedge 1 \leq y \leq 2)) \stackrel{\varepsilon}{\Leftrightarrow\!\!\rightarrow} \\
(P, (x \leq 2 \wedge 1 \leq y \leq 4 \wedge 1 \leq y \Leftrightarrow x \leq 2)) \stackrel{\tau}{\Leftrightarrow\!\!\rightarrow} \\
(Q, (x < 1 \wedge 1 \leq y \Leftrightarrow x \leq 2 \wedge 1 \leq y < 3)) \stackrel{\varepsilon}{\Leftrightarrow\!\!\rightarrow} (Q, (1 \leq y \Leftrightarrow x \leq 2))
\end{array}
$$

Thus, we have that $(X, (x = y = 0)) \Leftrightarrow\!\!\rightarrow^\star (Q, (1 \leq y \Leftrightarrow x < 2))$.

The following theorem shows that the symbolic semantics of processes is a full characterisation of the concrete semantics.

Theorem 3.1 (Correctness of Symbolic Semantics) *Assume* P^0, $P^f \in \mathcal{P}$, D^0, $D^f \in \mathcal{B}(\mathcal{C})$, *and* u^0, u^f *are clock assignments.*

- *(Soundness) whenever* $(P^0, \{u^0\}) \Leftrightarrow\!\!\rightarrow^\star (P^f, D^f)$ *then* $(P^0, u^0) \rightsquigarrow^\star (P^f, u^f)$ *for all* $u^f \in D^f$
- *(Completeness) whenever* $(P^0, u^0) \rightsquigarrow^\star (P^f, u^f)$ *then* $(P^0, \{u^0\}) \Leftrightarrow\!\!\rightarrow^\star (P^f,\ D^f)$ *for some* D^f *and* $u^f \in D^f$

where $\{u^0\}$ *denotes the clock constraint with* u^0 *being the only solution.*

Proof: We prove both soundness and completeness by induction on the length of transition sequences. First, we show that all transition sequences $(P^0, u^0) \rightsquigarrow^n (P^n, u^n)$ can be assumed (without loss of generality) to have alternating actions [4] ,

[4] This follows from the Time Continuity Lemma (see e.g. [Yi91]).

i.e. the form $(P^0,u^0) \overset{d_1}{\rightsquigarrow} (P^0,u^1) \overset{a_2}{\rightsquigarrow} (P^2,u^2) \overset{d_3}{\rightsquigarrow} \ldots \overset{a_{n-1}}{\rightsquigarrow} (P^{n-1},u^{n-1}) \overset{d_n}{\rightsquigarrow} (P^{n-1},u^n)$, $d_i \in I\!R_+$ and $a_i \in \mathcal{A}ct$. Neighboring action-transition in the form $\ldots \overset{a_i}{\rightsquigarrow} (P^i,u^i) \overset{a_{i+1}}{\rightsquigarrow} \ldots$ can be transformed to $\ldots \overset{a_i}{\rightsquigarrow} (P^i,u^i) \overset{0}{\rightsquigarrow} (P^i,u^i) \overset{a_{i+1}}{\rightsquigarrow} \ldots$ by inserting 0-delay transitions, and neighboring delay-transitions in the form $\ldots \overset{d_i}{\rightsquigarrow} (P^i,u^i) \overset{d_{i+1}}{\rightsquigarrow} (P^i,u^{i+1}) \ldots$ can be transformed to $\ldots \overset{d_i+d_{i+1}}{\rightsquigarrow} (P^i,u^{i+1}) \ldots$.

(Soundness) Assume $(P^0,\{u^0\}) \leftrightarrows^n (P^n,D^n) \overset{\sigma}{\leftrightarrows} (P^{n+1},D^{n+1})$. By induction $(P^0,u^0) \rightsquigarrow^n (P^n,u^n)$ and for all $u^n \in D^n$. We need to prove for all $u^{n+1} \in D^{n+1}$, $(P^n,u^n) \overset{\sigma}{\rightsquigarrow} (P^{n+1},u^{n+1})$ for some $u^n \in D^n$. There are two cases since $\sigma \in (\mathcal{A}ct \cup \{\varepsilon\})$:

- $(\sigma \in \mathcal{A}ct)$ From $(P^n,D^n) \overset{a}{\leftrightarrows} (P^{n+1},D^{n+1})$ and Definition 3.2 we have $P^n \overset{gar}{\leftrightarrows} P^{n+1}$ and $D^{n+1} = r(g \wedge D^n)$. Further, due to Definition 3.1, we have $D^{n+1} = \{r[u] \mid u \in D^n \text{ and } g(u)\}$. For state (P^n,u^n), by Definition 2.4 and $P^n \overset{gar}{\leftrightarrows} P^{n+1}$, we get a transition $(P^n,u^n) \overset{a}{\rightsquigarrow} (P^{n+1},u^{n+1})$ such that $g(u^n)$. Thus, we have that for all $u^{n+1} \in D^{n+1}$, there is a $u^n \in D^n$ such that $g(u^n)$ and $u^{n+1} = r[u^n]$.
- $(\sigma = \varepsilon)$ From $(P^n,D^n) \overset{\varepsilon}{\leftrightarrows} (P^{n+1},D^{n+1})$ and Definition 3.2 we get $P^n = P^{n+1}$, $D^{n+1} = D^{n\uparrow P^n} = ((D^n \wedge I(P^n))^\uparrow \wedge I(P^n))$. Due to Definition 3.1 $D^{n+1} = \{u^n \oplus d \mid u^n \in D^n \wedge d \in I\!R_+ \wedge I(P^n)(u^n) \wedge I(P^n)(u^n \oplus d)\}$. For (P^n,u^n), by Definition 2.4, we get $(P^n,u^n) \overset{d}{\leftrightarrows} (P^n,u^n \oplus d)$ if $I(u^n)$ and $I(u^n \oplus d)$. Thus, for all $u^{n+1} \in D^{n+1}$, there is a $u^n \in D^n$ such that $I(u^n)$, $I(u^{n+1})$ and $u^{n+1} = u^n + d$.

(Completeness) Assume $(P^0,u^0) \rightsquigarrow^n (P^n,u^n) \overset{\sigma}{\rightsquigarrow} (P^{n+1},u^{n+1})$. By induction step $(P^0,\{u^0\}) \leftrightarrows^n (P^n,D^n)$ and $u^n \in D^n$. We need to prove $(P^n,D^n) \overset{\sigma}{\leftrightarrows} (P^{n+1},D^{n+1})$ for some D^{n+1} and $u^{n+1} \in D^{n+1}$. There are two cases since $\sigma \in (\mathcal{A}ct \cup I\!R_+)$:

- $(\sigma \in \mathcal{A}ct)$ From $(P^n,u^n) \overset{a}{\rightsquigarrow} (P^{n+1},u^{n+1})$ and Definition 2.4 it follows that $P^n \overset{gar}{\leftrightarrows} P^{n+1}$, $u^{n+1} = r[u^n]$ and $g(u^n)$. By Definition 3.2 and $P^n \overset{gar}{\leftrightarrows} P^{n+1}$ we get $(P^n,D^n) \overset{a}{\leftrightarrows} (P^{n+1},D^{n+1})$ and $D^{n+1} = r(D^n \wedge g)$. Due to Definition 3.1 $r(D^n \wedge g) = \{r[u] \mid u \in D^n \wedge g(u)\}$. Thus, $r[u^n] \in D^{n+1}$, that is $u^{n+1} \in D^{n+1}$.
- $(\sigma \in I\!R_+)$ From $(P^n,u^n) \overset{d}{\rightsquigarrow} (P^{n+1},u^{n+1})$, $d \in I\!R_+$ and Definition 2.4 we have $P^n = P^{n+1}$, $u^{n+1} = u^n \oplus d$, $I(P^n)(u^n)$, and $I(P^n)(u^n \oplus d)$. From Definition 3.1 and 3.2 we get $(P^n,D^n) \overset{\varepsilon}{\leftrightarrows} (P^n,D^{n+1})$ and $D^{n+1} = D^{n\uparrow P^n} = \{u \oplus e \mid I(P^n)(u) \wedge I(P^n)(u \oplus e) \wedge u \in D^n \wedge e \in I\!R_+\}$. From $u^n \in D^n$, $I(P^n)(u^n)$, $I(P^n)(u^n \oplus d)$ and $d,e \in I\!R_+$ we have that $u^n \oplus d \in D^{n+1}$, that is $u^{n+1} \in D^{n+1}$. □

4 Finite Symbolic Semantics of Processes

While the symbolic semantics of timed automata defined in the previous section is coarser than the concrete semantics, it is not a suitable base for a verification algorithm because it is still infinite. The problem is that the symbolic state-space of a timed automaton is not guaranteed to be finite as the number of clock constraints in each symbolic state is unbounded.

In this section we shall develop a symbolic semantics which is guaranteed to yield a finite number of so-called *normalised* symbolic states, but still fully characterises

the concrete semantics. Normalisation will be used in the next section to develop a verification algorithm for processes. The idea of normalisation is based on the region graph technique [AD90, ACD90] and similar solutions have been proposed in [Rok93, WT94, DT98].

Example 4.1 *Reconsider the timed automaton A_X of Figure 2. It has the following infinite symbolic transition sequence:*

$$
\begin{array}{l}
(X,(x=y=0)) \stackrel{\varepsilon}{\Leftrightarrow\!\!\rightarrow} (X,(x=y \wedge y \leq 1)) \stackrel{\tau}{\Leftrightarrow\!\!\rightarrow} \\
(P,(x=y \wedge y \leq 1)) \stackrel{\varepsilon}{\Leftrightarrow\!\!\rightarrow} \\
(P,(x=y \wedge y \leq 2)) \stackrel{a}{\Leftrightarrow\!\!\rightarrow} \\
(P,(x=0 \wedge 1 \leq y \leq 2)) \stackrel{\varepsilon}{\Leftrightarrow\!\!\rightarrow} \\
(P,(x \leq 2 \wedge 1 \leq y \leq 4 \wedge 1 \leq y \Leftrightarrow x \leq 2)) \stackrel{a}{\Leftrightarrow\!\!\rightarrow} \\
(P,(x=0 \wedge 2 \leq y \leq 4 \wedge 2 \leq y \Leftrightarrow x \leq 4)) \stackrel{\varepsilon}{\Leftrightarrow\!\!\rightarrow} \\
(P,(x \leq 2 \wedge 2 \leq y \leq 6 \wedge 2 \leq y \Leftrightarrow x \leq 4)) \stackrel{a}{\Leftrightarrow\!\!\rightarrow} \\
(P,(x=0 \wedge 3 \leq y \leq 6 \wedge 3 \leq y \Leftrightarrow x \leq 6)) \stackrel{\varepsilon}{\Leftrightarrow\!\!\rightarrow} \\
(P,(x \leq 2 \wedge 3 \leq y \leq 8 \wedge 3 \leq y \Leftrightarrow x \leq 6)) \stackrel{a}{\Leftrightarrow\!\!\rightarrow} \ldots
\end{array}
$$

Clearly, the transition sequence in Example 4.1 yields an infinite number of symbolic states as the number of clock constraints is unbounded in the control-node P. The source of the problem is the upper bound on the y-clock that is gradually increased in the symbolic states of the sequence. The key question is how to prevent clock bounds from growing infinitely, without changing the semantics of the processes.

Normalisation is based on the observation that there is a *maximal constant* $k = 2$ appearing in the guards and location invariants of the automaton A_X. When a clock value have grown beyond 2 in A_X, the exact value does not matter anymore since it can not be distinguished by reachability analysis if the clock values over 2 are not of interest. It follows that a given symbolic state (P, D) of A_X can be extended to include *all* clock values that can not be distinguished from the ones already in D, in the above sense.

4.1 Normalisation of Clock Constraints

In this section we define a *normalisation* operator for clock constraints, which is parametrised with a given constant m. We first need to study the syntactical representation of clock constraints more carefully.

A well-known way to represent the class of constraints $\mathcal{B}(\mathcal{C})$ is to use difference bounded matrices (DBM, see [Bel57, Dil89, BL96]). A DBM is a matrix representation providing a canonical representation of clock constraints. For the purposes of this paper, it suffices to consider a DBM D as a constraint in the form $D = \bigwedge_{j \neq i}^{n} x_i \Leftrightarrow x_j \leq d_{ij}$, where $x_0 = 0$, and d_{ij} are integer numbers[5]. In general, there are several DBM's describing the same set of solutions. However, it has been shown that for each $D \in \mathcal{B}(\mathcal{C})$ there is a unique DBM, denoted D^C, with the same solution set as D,

[5]For reasons of simplicity and clarity in presentation we will only consider the non-strict orderings in the remainder of the paper. However, the techniques given extends easily to strict orderings.

which is *closed under entailment* in the sense that each d_{ij} can not be strengthened without reducing its solution set [Bel57, Dil89].

We now define a normalisation operator on the DBM-representation of clock constraints.

Definition 4.1 *Assume $D \in \mathcal{B}(\mathcal{C})$ and a natural number m. The m-normalisation of D, denoted $\mathsf{norm}_m(D)$ is the clock constraint obtained by substituting in D^C: all $d_{ij} > m$ with ∞, all $d_{ij} < \Leftrightarrow m$ with $\Leftrightarrow m$, and let all d_{ij} with $|d_{ij}| \leq m$ remain d_{ij}.*

Consider again the solution set A for the case of two clocks, shown in Figure 1. The operation $\mathsf{norm}_m(D)$ is illustrated in Figure 1(d). Intuitively, $\mathsf{norm}_m(D)$ is the clock constraint D where all upper bounds greater than m are removed and all lower bounds greater than m are replaced with m.

4.2 Normalised Symbolic Transition Rules

In Section 3.2 we have defined a symbolic semantics of processes, which is based on partitioning the concrete state-space in terms of the clock constraints $\mathcal{B}(\mathcal{C})$. In this section we shall give a *finite* symbolic semantics of processes which is based the normalisation operator on clock constraints. Let $\mathcal{B}_m(\mathcal{C})$ denote the subset of $\mathcal{B}(\mathcal{C})$ with no constants greater than m. We shall see that for a given constant m the normalised state-space is partitioned in terms of the constraints $\mathcal{B}_m(\mathcal{C})$ rather than $\mathcal{B}(\mathcal{C})$. As there are finitely many constraints in $\mathcal{B}_m(\mathcal{C})$ and finitely many control-nodes for a given process, the normalised symbolic state-space is guaranteed to be finite.

Let $\mathrm{M}(P)$ denote the maximal integer constant that appears in the guards and the location invariants of P. For a given timed automaton P and a constant $m > \mathrm{M}(P)$, we define the *normalised symbolic semantics* as a transition system where the set of states are the symbolic states (P, D) with $D \in \mathcal{B}_m(\mathcal{C})$. The normalised transition relation $\Longrightarrow_m$ is defined as follows:

Definition 4.2 (Normalised Symbolic Transition Rules) *Assume the symbol ε representing delays, $\sigma \in \mathcal{A}ct \cup \{\varepsilon\}$ and m is a natural number. We define*

$$(P, D) \stackrel{\sigma}{\Longrightarrow}_m (P', \mathsf{norm}_m(D')) \quad \textit{if} \quad (P, D) \stackrel{\sigma}{\Leftrightarrow\!\!\!\!\rightarrow} (P', D')$$

We shall write $(P^0, D^0) \Longrightarrow_m^n (P^n, D^n)$ to denote an alternating sequence in the form: $(P^0, D^0) \stackrel{\varepsilon}{\Longrightarrow}_m (P^1, D^1) \stackrel{a_2}{\Longrightarrow}_m (P^2, D^2) \stackrel{\varepsilon}{\Longrightarrow}_m \ldots \stackrel{a_{n-1}}{\Longrightarrow}_m (P^{n-1}, D^{n-1}) \stackrel{\varepsilon}{\Longrightarrow}_m (P^n, D^n)$ for $a_i \in \mathcal{A}ct$, and $(P^0, D^0) \Longrightarrow_m^\star (P^f, D^f)$ if $(P^0, D^0) \Longrightarrow_m^n (P^n, D^n)$ for some n. □

Thus, the only difference between the normalised symbolic semantics and the symbolic semantics, defined in Definition 3.2, is that the clock constraints are normalised in the normalised semantics. As all m-normalised constraints belong to $\mathcal{B}_m(\mathcal{C})$, and $\mathcal{P}$ is finite, the normalised symbolic semantics is guaranteed to yield a finite number of symbolic states.

Example 4.2 *Reconsider the infinite symbolic transition sequence shown in Example 4.1. In the finite symbolic semantics, with $m = 3$, we get the following transition sequence:*

$$
\begin{array}{l}
(X,(x=y=0)) \stackrel{\varepsilon}{\Longrightarrow}_m (X,(x=y\wedge y<1)) \stackrel{\tau}{\Longrightarrow}_m \\
(P,(x=y\wedge y<1)) \stackrel{\varepsilon}{\Longrightarrow}_m \\
(P,(x=y\wedge y<2)) \stackrel{a}{\Longrightarrow}_m \\
(P,(x=0\wedge 1\leq y<2)) \stackrel{\varepsilon}{\Longrightarrow}_m \\
(P,(x<2\wedge 1\leq y\wedge 1\leq y \Leftrightarrow x<2)) \stackrel{a}{\Longrightarrow}_m \\
(P,(x=0\wedge 2\leq y\wedge 2\leq y \Leftrightarrow x)) \stackrel{\varepsilon}{\Longrightarrow}_m \\
(P,(x\leq 2\wedge 2\leq y\wedge 2\leq y \Leftrightarrow x)) \stackrel{a}{\Longrightarrow}_m \\
(P,(x=0\wedge 3\leq y\wedge 3\leq y \Leftrightarrow x)) \stackrel{\varepsilon}{\Longrightarrow}_m \\
(P,(x\leq 2\wedge 3\leq y\wedge 3\leq y \Leftrightarrow x)) \stackrel{a}{\Longrightarrow}_m \ldots
\end{array}
$$

which is the sequence of Example 4.1 but with all clock constraints normalised.

To establish the correctness of the normalised symbolic semantics we shall need the following properties of the normalisation operator.

Lemma 4.1 *Assume $D \in \mathcal{B}(\mathcal{C})$ and a natural number m. We have that for all time assignments u with $\max_i(\lceil u(x_i)\rceil) = k$ and $k < m$*

(1) *$u \in D$ if and only if $u \in \mathsf{norm}_m(D)$*

(2) *$u \in (g \wedge D)$ if and only if $u \in (g \wedge \mathsf{norm}_m(D))$*

(3) *$u \in D^{\uparrow}$ if and only if $u \in (\mathsf{norm}_m(D))^{\uparrow}$*

(4) *$u \in \{x\}D$ if and only if $u \in \{x\}(\mathsf{norm}_m(D))$*

where $g \in \mathcal{B}_k(\mathcal{C})$ and $x \in \mathcal{C}$.

PROOF:

(1) Follows from the fact that the normalisation operator defined in Definition 4.1 does not affect any time assignments $u \in D^C$ with $\max_i(\lceil u(x_i)\rceil) < m$.

(2) Follows from (1) and the definition of the constraint operation "$\wedge$" in Definition 3.1.

(3) ($\Rightarrow$) This direction follows from the monotonicity of the $\uparrow$-operator and (1).
($\Leftarrow$) Assume $u \in (\mathsf{norm}_m(D))^{\uparrow}$. By Definition 3.1 there must exist $u = u' \oplus d$ for $u' \in \mathsf{norm}_m(D)$ and $d \in \mathbb{R}_+$. It follows that $\max_i(\lceil u(x_i)\rceil) < m$ since $\max_i(\lceil u'(x_i)\rceil) < m$, and due to (1) that $u' \in D$ which gives that $u' \oplus d \in D^{\uparrow}$.

(4) ($\Rightarrow$) This follows from the monotonicity of the reset-operator and (1).
($\Leftarrow$) Assume $u \in \{x\}(\mathsf{norm}_m(D))$. By Definition 3.1 we have that $u(x) = 0$ and that there exists $u' \in (\mathsf{norm}_m(D))$ with $u(y) = u'(y)$ for all $y \neq x$. There are now two cases, either $u'(x) < m$ or $u'(x) \geq m$. First assume $u'(x) < m$. Then $u' \in D$ due to (1) and it follows that $u \in \{x\}D$. For the case $u'(x) \geq m$, assume that there is no $u'' \in D$ with $u''(x) \geq m$ and $u'(y) = u''(y)$ for all $y \neq x$. Then $x \leq l$ for some $l < m$ must be a constraint in D^C. It follows from Definition 4.1 that $x \leq l$ must also be a constraint in $\mathsf{norm}_m(D)^C$ which contradicts the assumption that $u'(x) \geq m$. □

The following theorem shows how the normalised symbolic semantics characterises the concrete operational semantics defined in the Section 2.2.

Theorem 4.1 (Correctness of Normalised Symbolic Semantics)
Assume $P^0 \in \mathcal{P}$, $k = \mathrm{M}(P^0)$ *and* u^0 *is a time assignment.*

- *(Soundness) whenever* $(P^0, \{u^0\}) \Longrightarrow^{\star}_{m} (P^f, D^f)$ *then* $(P^0, u^0) \leadsto^{\star} (P^f, u^f)$ *for all* $u^f \in D^f$ *and* $m > \max(k, \max_i(\lceil u^f(x_i) \rceil))$
- *(Completeness) whenever* $(P^0, u^0) \leadsto^{\star} (P^f, u^f)$ *and* $m > \max(k, \max_i(\lceil u^f(x_i) \rceil))$ *then* $(P^0, \{u^0\}) \Longrightarrow^{\star}_{m} (P^f, D^f)$ *for some* D^f *and* $u^f \in D^f$

where $\{u^0\}$ *denotes the clock constraint with* u_0 *being the only solution.*

Proof: We prove both soundness and completeness by induction on the length of the transition sequences.

(Soundness) Assume $(P^0, \{u^0\}) \Longrightarrow^{n}_{m} (P^n, D^n) \overset{\sigma}{\Longrightarrow}_{m} (P^{n+1}, D^{n+1})$. By induction we have $(P^0, u^0) \leadsto^{n} (P^n, u^n)$ for all $u^n \in D^n$ with $\max_i(\lceil u^n(x_i) \rceil) < m$, and we need to prove, for all $u^{n+1} \in D^{n+1}$ with $\max_i(\lceil u^{n+1}(x_i) \rceil) < m$, $(P^n, u^n) \overset{\sigma}{\leadsto} (P^{n+1}, u^{n+1})$, for some $u^n \in D^n$ such that $\max_i(\lceil u^n(x_i) \rceil) < m$. We have two cases since $\sigma \in \mathcal{A}ct$ or $\sigma = \varepsilon$:

- $(\sigma \in \mathcal{A}ct)$ From $(P^n, D^n) \overset{\sigma}{\Longrightarrow}_{m} (P^{n+1}, D^{n+1})$, Definition 4.2 and 3.2 we have $P^n \overset{g\,a\,r}{\Leftrightarrow} P^{n+1}$ and $D^{n+1} = \mathsf{norm}_m(r(g \wedge D^n))$. Due to Definition 3.1, we get $D^{n+1} = \mathsf{norm}_m(\{r[u^n] \mid u^n \in D^n \text{ and } g(u^n)\})$. Further, by Lemma 4.1(1) and $\max_i(\lceil u^{n+1}(x_i) \rceil) < m$ we have $u^{n+1} \in \{r[u^n] \mid u^n \in D^n \text{ and } g(u^n)\}$. From Definition 2.4, Lemma 4.1(2), 4.1(4) and $P^n \overset{g\,a\,r}{\Leftrightarrow} P^{n+1}$ we get a transition $(P^n, u^n) \overset{a}{\leadsto} (P^{n+1}, r[u^n])$ such that $g(u^n)$ and $\max_i(\lceil u^n(x_i) \rceil) < m$. Thus, we have that for all $u^{n+1} \in D^{n+1}$ with $\max_i(\lceil u^{n+1}(x_i) \rceil) < m$ there is a $u^n \in D^n$ with $\max_i(\lceil u^n(x_i) \rceil) < m$ such that $g(u^n)$ and $u^{n+1} = r[u^n]$.
- $(\sigma = \varepsilon)$ From $(P^n, D^n) \overset{\varepsilon}{\Longrightarrow}_{m} (P^{n+1}, D^{n+1})$, Definition 4.2 and 3.2 we get $P^n = P^{n+1}$ and $D^{n+1} = \mathsf{norm}_m(D^{n\uparrow P^n})$. By Lemma 4.1 and $\max_i(\lceil u^{n+1}(x_i) \rceil) \leq k$ we have $u^{n+1} \in D^{n\uparrow P^n} = (D^n \wedge I(P^n))^{\uparrow} \wedge I(P^n)$, which by Definition 3.1 has the solution set $(\{u^n \oplus d \mid u^n \in D^n \wedge d \in \mathbb{R}_+ \wedge I(P^n)(u^n) \wedge I(P^n)(u^n \oplus d)\})$. By Definition 2.4, Lemma 4.1(3) and (P^n, u^n) we get $(P^n, u^n) \overset{d}{\Leftrightarrow} (P^n, u^n \oplus d)$ such that $I(u^n)$, $I(u^n \oplus d)$ and $\max_i(\lceil u^n(x_i) \rceil) < m$. Thus, for all $u^{n+1} \in D^{n+1}$ with $\max_i(\lceil u^{n+1}(x_i) \rceil) < m$, there is a $u^n \in D^n$ with $\max_i(\lceil u^n(x_i) \rceil) < m$ such that $I(u^n)$, $I(u^{n+1})$ and $u^{n+1} = u^n \oplus d$ for some d.

(Completeness) Assume $(P^0, u^0) \leadsto^{n} (P^n, u^n) \overset{\sigma}{\leadsto} (P^{n+1}, u^{n+1})$ and $\max_i(\lceil u^{n+1}(x_i) \rceil) < m$. By induction step $(P^0, \{u^0\}) \Longrightarrow^{n}_{m} (P^n, D^n)$ and $u^n \in D^n$. We need to prove $(P^n, D^n) \overset{\sigma}{\Longrightarrow}_{m} (P^{n+1}, D^{n+1})$, for some D^{n+1} and $u^{n+1} \in D^{n+1}$. However, from Theorem 3.1 we have that there is a symbolic transition $(P^n, D^n) \overset{\sigma}{\Leftrightarrow} (P^{n+1}, D_s^{n+1})$ such that $u^{n+1} \in D_s^{n+1}$. Further, from Definition 4.2 we have $(P^n, D^n) \overset{\sigma}{\Longrightarrow}_{m} (P^{n+1}, D^{n+1})$ and $D^{n+1} = \mathsf{norm}_m(D_s^{n+1})$. By Definition 4.1 we have $D^{n+1} \supseteq D_s^{n+1}$ and thus $u^{n+1} \in D^{n+1}$. □

5 Checking Safety Properties of Processes

The language developed in Section 2 can be used to construct abstract models of existing systems or systems to be designed. In this section we discuss how to verify properties of such systems in terms of their abstract models.

5.1 Reachability Analysis

It has been pointed out that the practical goal of verification of real-time systems, in particular safety-critical systems, is often to verify safety properties [Hal93]. These properties are usually formalised as invariant properties in the form $\mathsf{INV}(\neg\varphi)$ read as "φ is invariantly false", where φ describes a certain undesired situation or logical property. For finite-state systems, safety properties can be verified simply by checking all reachable states whether they satisfy φ or not, that is by "reachability analysis".

Let $\mathrm{N}(D)$ denote the the maximal integer constants that appears in the constraints of D. We shall consider the following reachability problem:

Definition 5.1 (Reachability of Normalised Symbolic States) *Assume* $P^0, P^f \in \mathcal{P}$, $D^0, D^f \in \mathcal{B}(\mathcal{C})$ *and* $k = \max(\mathrm{M}(P^0), \mathrm{N}(D^f))$. *We say that a symbolic state* (P^f, D^f) *is* reachable *from* (P^0, D^0) *if for all* $m < k$, $(P^0, D^0) \Longrightarrow^{\star}_{m} (P^f, D)$ *for some* $D \in \mathcal{B}_m(\mathcal{C})$ *and* $D \wedge D^f \neq \emptyset$. □

5.2 An Algorithm for Reachability Analysis

In this section we present an algorithm for forwards reachability analysis[6] of timed automata based on the finite symbolic semantics. To improve the presentation, we shall simply call (P, D) a state instead of a (normalised) symbolic state whenever it is not confusing.

The algorithm is based on the following idea: Assume that we want to decide whether (P, D) may reach (P', D') in one step (i.e. without passing other control-nodes) or not. The first thing to check is whether it is possible for P to switch to P' directly. If this is not the case, i.e. $P \overset{g\,a\,r}{\Longleftrightarrow} P'$ for no P', g, a, r, we can immediately conclude that (P', D') is not reachable from (P, D) in one step. Now, assume $P \overset{g\,a\,r}{\Longleftrightarrow} P'$. To reach (P', D'), there should be clock constraints D^1, D^2 and D^3 such that $D' \wedge D^3 \neq \emptyset$ and $(P, D) \overset{\varepsilon}{\Longrightarrow}_m (P, D^1) \overset{a}{\Longrightarrow}_m (P', D^2) \overset{\varepsilon}{\Longrightarrow}_m (P', D^3)$. Note that D and D' are given. From the normalised symbolic semantics in Definition 4.2 we get $D^1 = \mathsf{norm}_m(D^{\uparrow P}), D^2 = \mathsf{norm}_m(r(g \wedge D^1)), D^3 = \mathsf{norm}_m(D^{2\uparrow P'})$. That is, $D^3 = \mathsf{norm}_m(\mathsf{norm}_m(r(g \wedge \mathsf{norm}_m(D)^{\uparrow P}))^{\uparrow P'})$.

The algorithm for forwards reachability analysis is shown in Figure 3. It uses two buffers for saving states: PASSED and WAITING, where PASSED holds the set of states that have been examined, and WAITING the set of states that are to be examined next. When the algorithm is started PASSED = {} and WAITING = $\{(P^0, D^{0\uparrow P^0})\}$, where $D^{0\uparrow P^0}$ is the largest set of clock assignments that can be reached by idling in P^0. The algorithm then repeatedly examines the states in WAITING. If a state (P, D)

[6] It can easily be adopted to backwards reachability analysis (see [YPD94]).

Input:
$P^0, P^f \in \mathcal{P}$, $D^0, D^f \in \mathcal{B}(\mathcal{C})$ and $m > \max(\mathrm{M}(P^0), \mathrm{N}(D^f))$.
Initial:
WAITING := $\{(P^0, D^{0\uparrow P^0})\}$
PASSED := $\{\}$
Repeat:
for all $(P, D) \in$ WAITING **do**
begin
if $D \not\subseteq D'$ for all $(P, D') \in$ PASSED **then**
begin
PASSED := PASSED $\cup \{(P, D)\}$ and
WAITING := WAITING $\cup \{(P', D')\}$ for all P', g, a, r
such that $P \overset{g\,a\,r}{\Leftrightarrow} P'$, $D' = \mathsf{norm}_m((r(g \wedge D))^{\uparrow P'})$
and $D' \neq \emptyset$.
end;
else WAITING := WAITING $\Leftrightarrow \{(P, D))\}$
end
until WAITING $= \emptyset$ or $(P^f, D') \in$ WAITING for some D' such that $D^f \wedge D' \neq \emptyset$.

Termination: if WAITING $= \emptyset$ **then** return "*no*" **else** return "*yes*".

Figure 3: An Algorithm for Reachability Analysis.

found in WAITING that is smaller[7] than a state (P, D') in PASSED, then (P, D) does not need to be examined further. Otherwise, put all the states that are reachable from (P, D) in one step into WAITING to be examined further, and put (P, D) into PASSED. The algorithm will terminate when WAITING is empty (i.e. nothing is left to be examined, and therefore fails to find the final state) or a state (P^f, D') is found, which includes a part of the final state (P^f, D^f) (i.e. $D^f \wedge D' \neq \emptyset$).

We now show that the algorithm is partially correct: Given proper inputs, it always provides the right answer.

Theorem 5.1 (Partial Correctness) *For all initial states (P^0, D^0) and final states (P^f, D^f):*

1. *whenever the algorithm terminates with answer "*yes*", there exists $(P^0, u^0) \in (P^0, D^0)$, $(P^0, u^0) \rightsquigarrow^\star (P^f, u^f)$ for some $(P^f, u^f) \in (P^f, D^f)$*
2. *whenever the algorithm terminates with answer "*no*", then for all $(P^0, u^0) \in (P^0, D^0)$, $(P^0, u^0) \rightsquigarrow^\star (P^f, u^f)$ for no $(P^f, u^f) \in (P^f, D^f)$*

[7]The symbolic state (P, D) is smaller than (P', D') if $P = P'$ and $D \subseteq D'$.

PROOF: Let $\mathrm{SP}_m((P,D))$ denote $\{\ (P',D') \mid (P,D) \Longrightarrow_m^\star (P',D')\ \}$, i.e. the set of all normalised symbolic states reachable from (P,D). We first show that the two sets PASSED and WAITING used in the algorithm satisfies the following invariant property:

$$\mathrm{SP}_m((P^0,D^0)) \;=\; \textsc{Passed} \;\cup \bigcup_{(P,D)\in \textsc{Waiting}} \mathrm{SP}_m((P,D))$$

(**Initial:**) Trivial, as $\mathrm{SP}_m((P^0,D^0)) = \mathrm{SP}_m((P^0,D^{0\uparrow P^0}))$, $\textsc{Passed} = \emptyset$, and $(P^0,D^0) \stackrel{\varepsilon}{\Longrightarrow}_m (P^0,D^{0\uparrow P^0})$ by Definition 3.2.
(**Repeat:**) Assume the invariant property holds for the current values of PASSED and WAITING, $(P,D) \in \textsc{Waiting}$, and $P \neq P^f$ or $D \wedge D^f = \emptyset$. Assume also that there is no $(P,D') \in \textsc{Passed}$ such that $D \subseteq D'$. The algorithm then updates PASSED and WAITING to $\textsc{Passed} \cup \{(P,D)\}$ and $\textsc{Waiting} \Leftrightarrow \{(P,D)\} \cup \{(P',D') \mid P \stackrel{g\,a\,r}{\Leftrightarrow} P'$ and $D' = \mathsf{norm}_m((r(D \wedge g))^{\uparrow P'})\}$. This does not modify the set of states on the r.h.s. of the invariant property, since the only symbolic state (P,D) removed from WAITING is added to PASSED and all successor states of (P,D) are added to WAITING.

Now assume $D \subseteq D'$ for some $(P,D') \in \textsc{Passed}$. Then $\mathrm{SP}_m((P,D)) \subseteq \mathrm{SP}_m((P,D'))$ and thus (P,D) does not have to be further explored.
(**Termination:**) Trivial as PASSED and WAITING are not updated. Note that if the algorithm terminates on the criteria $\textsc{Waiting} = \emptyset$, then $\mathrm{SP}_m((P^0,\ D^0)) = \textsc{Passed}$, i.e. PASSED holds all reachable normalised symbolic states.

We now establish that the criteria (1) and (2) for partial correctness holds.

(1) The algorithm terminates with "*yes*" whenever there exists $(P,D) \in \textsc{Waiting}$, $P = P^f$ and $D \wedge D^f \neq \emptyset$. From the above invariant we have $(P^0,D^0) \Longrightarrow_m^\star (P^f,D)$. It follows from Theorem 4.1 that $(P^0,u^0) \rightsquigarrow^\star (P^f,u^f)$ for some $u^0 \in D^0$ and $u^f \in D^f$.

(2) When the algorithm terminates with "*no*" $\textsc{Passed} = \mathrm{SP}_m((P^0,D^0))$ and no $(P,D) \in \textsc{Passed}$ is found such that $P = P^f$ and $D \wedge D^f \neq \emptyset$. By Theorem 4.1 we have that $(P_0,u_0) \rightsquigarrow^\star (P_f,u_f)$ for no $u^0 \in D^0$ and $u^f \in D^f$. □

Finally, we prove total correctness of the algorithm: Given proper inputs, it always terminates with an answer.

Theorem 5.2 (Total Correctness) *For all initial states (P^0,D^0) and final states (P^f,D^f), the algorithm always terminates with an answer which is either "*yes*" or "*no*".*

PROOF: The theorem follows from the following two lemmas.

Lemma 5.1 (Closure Property of Clock Constraints) *Assume that $\mathcal{C}$ is a set of clocks, $x \in \mathcal{C}$, and $D, D' \in \mathcal{B}(\mathcal{C})$ with solution sets A and A'. Then there exist clock constraints D^1, D^2, $D^3 \in \mathcal{B}(\mathcal{C})$ with solution sets $A \wedge A'$, $A^\uparrow$, and $\{x\}A$*

PROOF: See [Dil89]. □

Lemma 5.2 *Assume $D \in \mathcal{B}(\mathcal{C})$ and a non-negative integer number m. Then* $\mathsf{norm}_m(D) \in \mathcal{B}_m(\mathcal{C})$*, where $\mathcal{B}_m(\mathcal{C})$ is the subset of $\mathcal{B}(\mathcal{C})$ with no constants greater than m.*

PROOF: Follows from Definition 4.1. □

Thus, due to Lemma 5.1 the class of clock constraints is closed under the operations on constraints, used in the algorithm. Further, from Lemma 5.2 the number of constraints manipulated by the algorithm is bounded. Since the number of control-nodes of P_0 is finite, we have that the algorithm is guaranteed to terminate with an answer. □

6 Examples

We have implemented the algorithm presented in the previous section in the two verification tools TAB and UPPAAL. The tool TAB, was developed in 1993 based on the symbolic and on-the-fly backwards reachability analysis algorithm for timed automata presented in the conference version of this paper [YPD94]. TAB is implemented in a constraint solver developed at the Swedish Institute of Computer Science (SICS) called Prolog Constraint Solver [Nil93].

In this section, we present two examples which have been analysed in the successor of TAB, called UPPAAL [LPY95, LPY97]. The first version of UPPAAL, implemented in C^{++} and with efficient operations on constraints, was finished in 1995 [LPY95]. In addition to clock variables the UPPAAL-model, which is based on the model of networks of timed automata presented in this paper, has integer data variables. These variables do not change their values at the delay-transitions as the clock variables do; they can only be assigned to values from finite domains, and therefore they will not cause infinite-stateness.

6.1 Fischer's Mutual Exclusion Protocol

The protocol was proposed originally by Fischer and described by Lamport [AL92]. It is to guarantee mutual exclusion in a concurrent system consisting of several processes using a variable shared among the processes. Each of the processes is assumed to have a local clock. The idea behind the protocol is that the timing constraints on the local clocks ensure that only one process can set the shared variable to its own process number, then later if the shared variable is still equal to its own number, enter the critical section.

Assume a concurrent system with n processes $P_1, \ldots, P_n$. We use x_i to model the local clock for each process P_i. The formal description of P_i is given in Figure 4, and the timed automaton described by P_i is illustrated in Figure 5 [8].

This is a simplified version of the original protocol and has been studied by researchers, e.g. [AL92, Sha93], which permits only one process to enter the critical section and never exits it. Recovery actions from failure to enter the critical section

[8] In Figure 5 and 7 we adopt the convention that when a transition is not labelled with an action, it means that the transition is an internal one, that is τ.

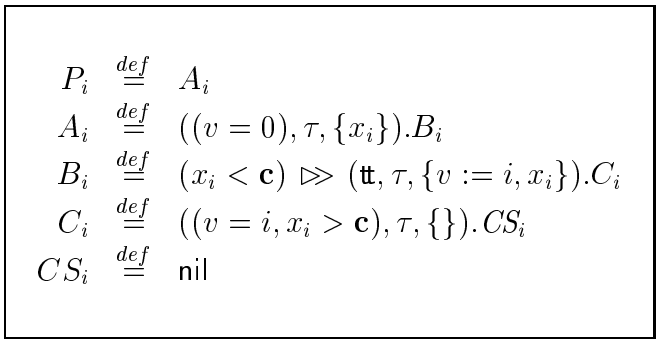


Figure 4: Formal Description of Fischer's Protocol.

are omitted. However, the protocol can be extended to an actual mutual exclusion algorithm.

The processes P_i may be in either of the four local states A_i, B_i, C_i, CS_i. Initially, all processes are in their A-states and the shared variable v is initially 0. A process, P_i, that tries to enter the critical section changes state from A_i to B_i if it sees $v = 0$. In B_i, it will move to C_i before the clock x_i proceeds to $\mathbf{c}$, and in doing so, reset the clock x_i (i.e. $x_i := 0$) and assign v to its own process number (i.e. $v := i$). From C_i, it can move to the critical section CS_i if v is still equal to its process number (i.e. $v = i$) when the clock value of x_i is larger than $\mathbf{c}$.

Intuitively, the protocol behaves as follows: The constraints on the shared variable v ensure that a process must reach B-node before any process reach C-node; otherwise, it will never move from A-node to B-node. The timing constraints on the clocks ensure that all processes in C-node must wait until all processes in B-node reach C-node. The last process that reached C-node and set v to its own process number gets the right to enter its critical section. In fact, the protocol will guarantee mutual exclusion for any non-zero constant $\mathbf{c}$.

We need to check that the mutual exclusion property is satisfied, i.e. there will never be more than one process which may reach the critical section, CS_i. The requirement can be formalised as follows: The concurrent system, with an initial state where the control-node is $A_1 \mid \ldots \mid A_n$ and arbitrary variable assignment, will never reach a state where the control-node is in the form

$$S_1 \mid \ldots \mid CS_k \mid \ldots \mid CS_l \mid \ldots \mid S_n$$

for some $k, l \leq n$ and $S_i \in \{A_i, B_i, C_i, CS_i\}$.

We have used UPPAAL and verified the system consisting of 12 processes and with $\mathbf{c} = 1$, which satisfies the property [9].

6.2 A Railway Control System

We consider a railway control system to automatically control trains passing a critical point such as a bridge. The idea is to use a computer to guide trains from several

[9] UPPAAL version 2.18.3 consumed 8376 seconds of CPU time and 265 MB of memory on a Pentium Pro 200 MHz machine running Redhat Linux 5.0.

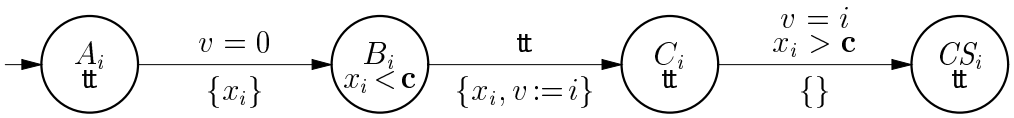


Figure 5: Fischer's Mutual Exclusion Protocol.

tracks crossing a single bridge instead of building many bridges. An obvious safety-property of such a system is to avoid the situation where more than one train are crossing the bridge at the same time.

Assume that the whole system consists of n trains and a controller. We model the system by the following process:

$$(C \mid \mathit{Train}_1 \mid \ldots \mid \mathit{Train}_n) \setminus A$$

where Train_i describe the behavior of the trains, C describes the behavior of the controller, and $A = \{\mathit{appr}_i, \mathit{stop}_i, \mathit{leave}_i, \mathit{go}_i\}$ is the set of internal channel names (or signals) between the trains and the controller.

To describe timing constraints, we use clocks y and x_i to model the local time of the controller and the trains respectively. The controller uses a list L for the trains waiting to cross the bridge. The formal descriptions of Train_i's and C are given in Figure 6 and illustrated in Figure 7.

Intuitively, when a train, Train_i, approaches the bridge it sends a signal to the controller within a certain distance. If the bridge is occupied the controller immediately sends a stop signal stop_i to prevent the train from entering the bridge. Otherwise, if the approaching train does not receive a stop signal within 10 time units, it will start to cross the bridge within 20 time units (but it will take at least 10 time units for a train to enter the bridge). The crossing train is assumed to leave the bridge within 3 to 5 time units; a stopped train will slow down and eventually stop after some delay. When the bridge is free again and the controller signals (by sending go_i) the first train in the waiting list to cross.

Assume that the system is started with the following control-node:

$$(\mathit{Free} \mid \mathit{Safe}_1 \mid \ldots \mid \mathit{Safe}_n)$$

and all clocks are initialised to 0.

We need to guarantee that the system will never reach a state where two trains are in node *Cross* (the clocks may have any values). That is, a state in the form:

$$(S_i \mid T_1 \mid \ldots \mid \mathit{Cross}_k \mid \ldots \mid \mathit{Cross}_l \mid \ldots \mid T_n)$$

for some $k, l \leq n$, $S_i \in \{\mathit{Free}, \mathit{Send}, \mathit{Occ}_1, \mathit{Occ}_2\}$ and $T_i \in \{\mathit{Safe}_i, \mathit{Appr}_i, \mathit{Slow}_i, \mathit{Stop}_i, \mathit{Start}_i\}$. We have verified that a system consisting of 6 trains satisfies the safety-requirement [10].

[10] UPPAAL version 2.18.3, installed on the same machine as in the previous example, consumed 143 MB of memory and 3019 seconds of CPU time.

$$
\begin{array}{rcl}
Train_i & \stackrel{def}{=} & Safe_i \\
Safe_i & \stackrel{def}{=} & (\mathtt{tt}, appr_i!, \{x_i\}).Appr_i \\
Appr_i & \stackrel{def}{=} & (x_i \leq 20) \rhd\!\!\rhd \big(((x_i \geq 0 \wedge x_i \leq 10), stop_i?, \{x_i\}).Slow_i \\
 & & \qquad + (x_i \geq 11), \tau, \{x_i\}).Cross_i \big) \\
Cross_i & \stackrel{def}{=} & (x_i \leq 5) \rhd\!\!\rhd ((x_i \geq 3), leave_i!, \{x_i\}).Safe_i \\
Slow_i & \stackrel{def}{=} & (x_i \leq 7) \rhd\!\!\rhd ((x_i \geq 5), \tau, \{x_i\}).Stop_i \\
Stop_i & \stackrel{def}{=} & (\mathtt{tt}, go_i?, \{x_i\}).Start_i \\
Start_i & \stackrel{def}{=} & (x_i \leq 15) \rhd\!\!\rhd ((x_i \geq 7), \tau, \{x_i\}).Cross_i
\end{array}
$$

$$
\begin{array}{rcl}
C & \stackrel{def}{=} & Free \\
Free & \stackrel{def}{=} & ((L = empty), appr_i?, \{L := i\}).Occ_1 \\
 & & + ((L \neq empty), \tau, \{n := hd(L), y\}).Send \\
Send & \stackrel{def}{=} & (y \leq 0) \rhd\!\!\rhd ((\mathtt{tt}, go_n!, \{\}).Occ_1 \\
Occ_1 & \stackrel{def}{=} & (\mathtt{tt}, leave_i?, \{L := L \Leftrightarrow i\}).Free \\
 & & + (\mathtt{tt}, appr_i?, \{n := i, y\}).Occ_2 \\
Occ_2 & \stackrel{def}{=} & (y \leq 0) \rhd\!\!\rhd (\mathtt{tt}, stop_n!, \{L := L :: n\}).Occ_1
\end{array}
$$

Figure 6: Formal Description of the Railway Control System.

7 Conclusion

We have presented an algebra of processes with clocks, which extends timed automata with algebraic operators. The algebra may serve as a formal description language for real-time communicating systems modelled as networks of timed automata. In particular, a parallel composition operator is introduced for timed automata to model communication and concurrency. The operators can be used to construct complex system descriptions in terms of simpler ones.

We have also presented a symbolic on-the-fly reachability analysis algorithm for the description language, based on constraint-solving techniques. The algorithm is proved to be sound (i.e. always provides the right answer) and complete (i.e. always terminates with an answer). It has been implemented in two automatic verification tools for checking safety properties of real-time systems: TAB and UPPAAL. In this paper, as examples, we apply UPPAAL to verify safety properties of a version of Fischer's mutual exclusion protocol and a railway control system.

There have been many proposals for verifying timed systems. However, most of them are intended to construct the whole state-space of a system or to obtain more efficient model-checking algorithms with respect to a real-time temporal logic, or to check equivalences between abstract specifications. We believe that the practical goal of verifying real-time systems, in particular safety-critical systems is to check

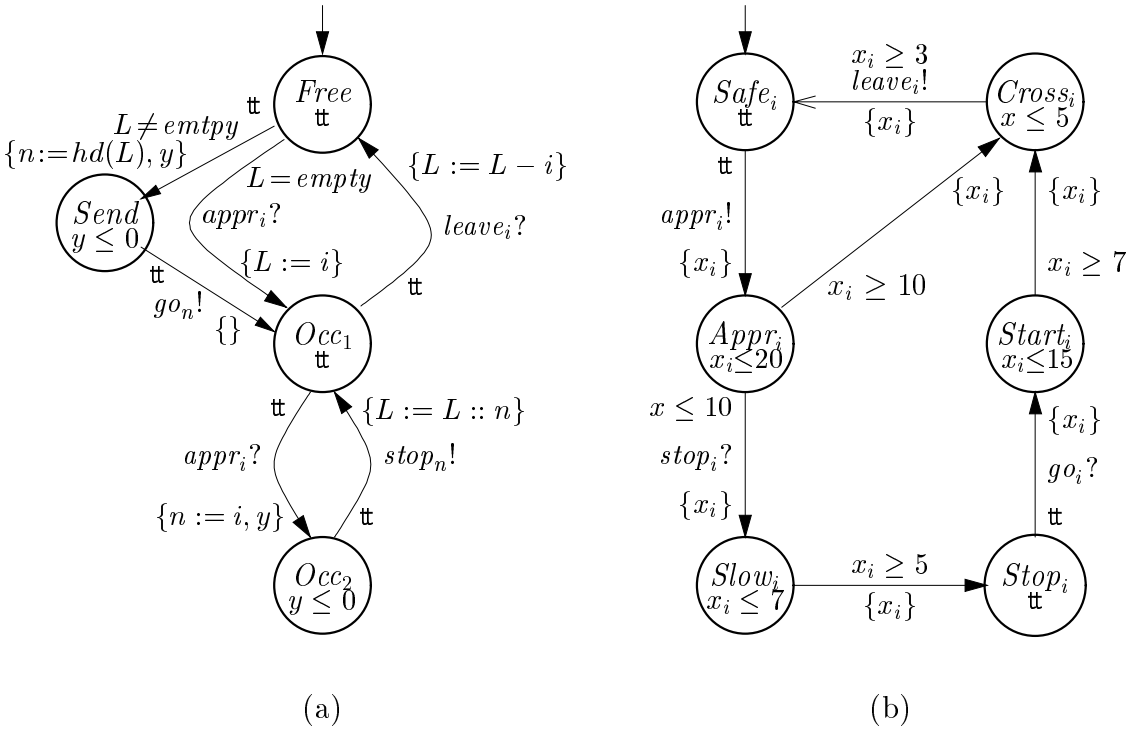


Figure 7: (a) Controller, (b) Train.

simple logical properties, which can be done without constructing the whole state-space. We are of the opinion that our approach is simpler as it is based directly on constraint-solving techniques and can be efficient in verifying systems consisting of many components as it explores (and generates) only the reachable part of the whole state-space.

References

[ACD90] Rajeev Alur, Costas Courcoubetis, and David Dill. Model-checking for Real-Time Systems. In *Proc. of Logic in Computer Science*, pages 414–425. IEEE Computer Society Press, June 1990.

[ACH+92] R. Alur, C. Courcoubetis, N. Halbwachs, D. Dill, and H. Wong-Toi. Minimization of Timed Transition System. In *Proc. of CONCUR '92, Theories of Concurrency: Unification an d Extension*, number 630 in Lecture Notes in Computer Science, 1992.

[AD90] Rajeev Alur and David Dill. Automata for Modelling Real-Time Systems. In *Proc. of Int. Colloquium on Algorithms, Languages and Programming*, number 443 in Lecture Notes in Computer Science, pages 322–335, July 1990.

[AL92] Martin Abadi and Leslie Lamport. An Old-Fashioned Recipe for Real Time. In *Proc. of REX Workshop "Real-Time: Theory in Practice"*, number 600 in Lecture Notes in Computer Science, 1992.

[Bel57] Richard Bellman. *Dynamic Programming.* Princeton University Press, 1957.

[BL96] Johan Bengtsson and Fredrik Larsson. UPPAAL a Tool for Automatic Verification of Real-time Systems. Master's thesis, Uppsala University, 1996. Available as `http://www.docs.uu.se/docs/rtmv/bl-report.pdf`.

[Cer92] Karlis Cerans. Decidability of Bisimulation Equivalences for Parallel Timer Processes. In *Proc. of CAV'92*, number 663 in Lecture Notes in Computer Science, Berlin, 1992. Springer–Verlag.

[CGL93] Karlis Cerans, Jens Chr. Godskesen, and Kim G. Larsen. Time Modal Specification – Theory and Tools. In *Proc. of the 5*th *Int. Conf. on Computer Aided Verification*, number 697 in Lecture Notes in Computer Science. Springer–Verlag, 1993.

[DB96] Pedro R. D'Argenio and Ed Brinksma. A Calculus for Timed Automata. In Bengt Jonsson and Joachim Parrow, editors, *Proc. of Formal Techniques in Real-Time and Fault-Tolerant Systems*, number 1135 in Lecture Notes in Computer Science, pages 110–129. Springer–Verlag, 1996.

[Dil89] David Dill. Timing Assumptions and Verification of Finite-State Concurrent Systems. In J. Sifakis, editor, *Proc. of Automatic Verification Methods for Finite State Systems*, number 407 in Lecture Notes in Computer Science, pages 197–212. Springer–Verlag, 1989.

[DT98] Conrado Daws and Stavros Tripakis. Model checking of real-time reachability properties using abstractions. In Bernard Steffen, editor, *Proc. of the 4*th *Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, number 1384 in Lecture Notes in Computer Science, pages 313–329. Springer–Verlag, 1998.

[Hal93] Nicolas Halbwachs. Delay Analysis in Synchronous Programs. In *Proc. of the 5*th *Int. Conf. on Computer Aided Verification*, number 697 in Lecture Notes in Computer Science, 1993.

[HLY92] U. Holmer, K.G. Larsen, and W. Yi. Decidability of bisimulation equivalence between regular timed processes. In *Proc. of Workshop on Computer Aided Verification*, number 575 in Lecture Notes in Computer Science, Berlin, 1992. Springer–Verlag.

[HNSY92] Thomas. A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic Model Checking for Real-Time Systems. In *Proc. of IEEE Symp. on Logic in Computer Science*, 1992.

[Hol91] Gerard Holzmann. *The Design and Validation of Computer Protocols.* Prentice Hall, 1991.

[LPY95] Kim G. Larsen, Paul Pettersson, and Wang Yi. Compositional and Symbolic Model-Checking of Real-Time Systems. In *Proc. of the 16*th *IEEE Real-Time Systems Symposium*, pages 76–87. IEEE Computer Society Press, December 1995.

[LPY97] Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a Nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1(1–2):134–152, October 1997.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice Hall, Englewood Cliffs, 1989.

[MT91] F. Moller and C. Tofts. Relating Processes with Respect to Speed. Technical Report ECS–LFCS–91–143, Department of Computer Science, University of Edinburgh, 1991.

[Nil93] Martin Nilsson. Piecewise Linear Constraints and Entailment. Technical report, Swedish Institute of Computer Science, August 1993.

[Rok93] Tomas Gerhard Rokicki. *Representing and Modeling Digital Circuits.* PhD thesis, Stanford University, 1993.

[Sha93] N. Shankar. Verification of Real-Time Systems Using PVS. In *Proc. of the 5*th *Int. Conf. on Computer Aided Verification*, number 697 in Lecture Notes in Computer Science. Springer–Verlag, 1993.

[WT94] Howard Wong-Toi. *Symbolic Approximations for Verifying Real-Time Systems.* PhD thesis, Stanford University, November 1994.

[Yi91] Wang Yi. *A Calculus of Real Time Systems.* PhD thesis, Department of Computer Science, Chalmers University of Technology, 1991.

[YPD94] Wang Yi, Paul Pettersson, and Mats Daniels. Automatic Verification of Real-Time Communicating Systems By Constraint-Solving. In Dieter Hogrefe and Stefan Leue, editors, *Proc. of the 7*th *Int. Conf. on Formal Description Techniques*, pages 223–238. North–Holland, 1994.

Paper B

Kim G. Larsen, Paul Pettersson, and Wang Yi. Compositional and Symbolic Model-Checking of Real-Time Systems. In *Proceedings of the 16*th *IEEE Real-Time Systems Symposium*, pages 76–87. IEEE Computer Society Press 1995.

Compositional and Symbolic Model-Checking of Real-Time Systems

Kim G. Larsen*, Paul Pettersson†, and Wang Yi†

* BRICS, Aalborg University, Denmark. Email: kgl@cs.auc.dk.
† Department of Computer Systems, Uppsala University, Sweden.
Email: {paupet,yi}@docs.uu.se.

Abstract. In this paper, we present *symbolic* and *compositional* techniques to avoid state explosion problems in model-checking for networks of timed automata. The symbolic technique is based on partitioning the inherently infinite state-space of timed automata in terms of simple constraints over the clocks variables instead of into regions. This allows the partitioning to take into account the particular system and the property to be verified. In practice, it will result in a considerably coarser and smaller partitioning than the property-independent region graph. The compositional *quotienting construction* allows properties of a network to be verified by gradually moving components from the system description (the network) into the specification that is a logical formula specifying the properties. This transforms the model-checking problem to validity-checking of logical formulas, which may terminate as soon as the formula becomes true or false without exploring the whole state-space of the network.

1 Introduction

Within the last decade model-checking has turned out to be a useful technique for verifying temporal properties of finite-state systems. Efficient model-checking algorithms for finite-state systems have been obtained with respect to a number of logics. However, the major problem in applying model-checking even to moderate-size systems is the potential combinatorial explosion of the state space arising from parallel composition. In order to avoid this problem, algorithms have been sought that avoid exhaustive state space exploration, either by *symbolic* representation of the states space using Binary Decision Diagrams [BCM+90], by application of *partial order* methods [GW91, Val90] which suppresses unnecessary interleavings of transitions, or by application of *abstractions* and *symmetries* [CFJ93, CGL92, EJ93].

In the last few years, model-checking has been extended to real-time systems, with time considered to be a dense linear order. A timed extension of finite automata through addition of a finite set of real-valued clock-variables has been put forward [AD94] (so-called timed automata), and the corresponding model-checking problem has been proven decidable for a number of timed logics including timed extensions of CTL (TCTL) [ACD90] and timed μ-calculus (T_μ) [HNSY94]. A state of a timed automaton is of the form (l, u), where l is a control-node and u is a clock-assignment holding the current values of the clock-variables . The crucial observation made by Alur, Courcoubetis and Dill and the foundation for decidability of model-checking is that the (infinite) set of clock-assignments may effectively be partitioned

into finitely many *regions* in such a way that clock-assignments within the same region induce states satisfying the same logical properties.

Model-checking of real-time systems based on the region technique suffers two potential types of explosion arising from parallel composition: *Explosion in the region space*, and *Explosion in the space of control-nodes*. We attack these problems by development and combination of two new verification techniques:

1. A *symbolic* technique reducing the verification problem to that of solving simple constraint systems (on clock-variables), and
2. A *compositional* quotient construction, which allows components of a real-time system to be gradually moved from the system into the specification. The intermediate specifications are kept small using minimisation heuristics.

The property-independent nature of regions leads to an extremely fine (and large) partitioning of the set of clock-assignments. Our symbolic technique allows the partitioning to take account of the particular property to be verified and will thus in practice be considerably coarser (and smaller).

For the explosion on control-nodes, recent work by Andersen [And95] on (untimed) finite-state systems gives experimental evidence that the quotient technique improves results obtained using Binary Decision Diagrams [BCM$^+$90] on some examples. Our aim of this paper is to make this new successful compositional model-checking technique applicable to real-time systems. For example, consider the following typical model-checking problem

$$\big(A_1 \mid \ldots \mid A_n\big) \models \varphi$$

where the A_i's are timed automata. We want to verify that the parallel composition of these satisfies the formula φ without having to construct the complete control-node space of $(A_1 \mid \ldots \mid A_n)$. We will avoid this complete construction by removing the components A_i one by one while simultaneously transforming the formula accordingly. Thus, when removing the component A_n we will transform the formula φ into the *quotient* formula $\varphi \,/\, A_n$ such that

$$\big(A_1 \mid \ldots \mid A_n\big) \models \varphi \quad \text{if and only if} \quad \big(A_1 \mid \ldots \mid A_{n-1}\big) \models \varphi \,/\, A_n$$

Now clearly, if the quotient is not much larger than the original formula we have succeeded in simplifying the problem. Repeated application of quotienting yields

$$\big(A_1 \mid \ldots \mid A_n\big) \models \varphi \quad \text{if and only if} \quad \mathbf{1} \models \varphi \,/\, A_n \,/ A_{n-1} \,/\, \ldots \,/\, A_1$$

where $\mathbf{1}$ is the unit with respect to parallel composition. However, these ideas alone are clearly not enough as the explosion may now occur in the size of the final formula instead. The crucial and experimentally "verified" observation by Andersen was that each quotienting should be followed by a minimisation of the formula based on a small collection of efficiently implementable strategies. In our setting, Andersen's collection is extended to include strategies for propagating and simplifying timing constraints.

Our new symbolic and compositional verification technique is developed for a real-time logic designed specifically for expressing safety and bounded liveness properties. Comparatively less expressive than TCTL and T_μ, the logic is still sufficiently expressive for practical purposes, and the logic allows a number of operators of other logics to be derived. Most importantly, the somewhat restrictive expressive power of our logic allows for extremely efficient model-checking.

For the logics TCTL and T_μ, [HNSY94] offers a symbolic verification technique. However, due to the high expressive power of these logics the partitioning employed in [HNSY94] is significantly finer (and larger) and implementation-wise more complicated than the symbolic technique we present in this paper. Our symbolic method is based on the constraint solving technique presented in [YPD94], where the technique was developed for simple reachability problems.

An initial effort in applying the compositional quotienting technique to real-time systems has been given in [LL95]. This work also contains experimental evidence of the potential benefits of the quotient technique in a real-time setting. However, being based directly on the (very fine) notion of regions, [LL95] suffers from a potential explosion in the region-space.

The outline of this paper is as follows: In the next section we give a short presentation of the notions of timed automata and networks; in section 3, the safety logic is presented and its expressive power is illustrated. Section 4 describes the symbolic verification technique based on constraint solving and section 5 describes the compositional quotienting technique. Both techniques are illustrated by an example. Section 6 concludes the paper.

2 Real-Time Systems

We shall use *timed transition systems* as a basic semantical model for real-time systems. The type of systems we are studying will be a particular class of timed transition systems that are syntactically described by *networks of timed automata* [YPD94, LL95].

2.1 Timed Transition Systems

A timed transition system is a labelled transition system with two types of labels: atomic actions and delay actions (i.e. positive reals), representing discrete and continuous changes of real-time systems.

Let $\mathcal{A}$ be a finite set of actions ranged over by a, b etc, and $\mathcal{P}$ be a set of atomic propositions ranged over by p, q etc. We use $\mathbb{R}_+$ to stand for the set of non-negative real numbers, Δ for the set of delay actions $\{\epsilon(d) \mid d \in \mathbb{R}_+\}$, and Σ for the union $\mathcal{A} \cup \Delta$.

Definition 2.1 (Timed Transition System) *A timed transition system over Σ and $\mathcal{P}$ is a tuple $\mathcal{S} = \langle S, s_0, \Leftrightarrow\!\!\!\rightarrow, V \rangle$, where S is a set of states, s_0 is the initial state, $\Leftrightarrow\!\!\!\rightarrow \subseteq S \times \Sigma \times S$ is a transition relation, and $V : S \to 2^{\mathcal{P}}$ is a proposition assignment function.* □

Note that the above definition is standard for labelled transition systems except that we introduced a proposition assignment function V, which for each state $s \in S$ assigns a set of atomic propositions $V(s)$ that hold in s.

In order to study compositionality problems we introduce a parallel composition between timed transition systems. Following [HL89] we suggest a composition parameterised with a synchronisation function generalising a large range of existing notions of parallel compositions. A *synchronisation function* f is a partial function $(\mathcal{A} \cup \{0\}) \times (\mathcal{A} \cup \{0\}) \hookrightarrow \mathcal{A}$, where 0 denotes a distinguished no-action symbol[1]. Now, let $\mathcal{S}_i = \langle S_i, s_{i,0}, \Leftrightarrow\!\!\rightarrow_i, V_i \rangle$, $i = 1, 2$, be two timed transition systems and let f be a synchronisation function. Then the *parallel composition* $\mathcal{S}_1 \mid_f \mathcal{S}_2$ is the timed transition system $\langle S, s_0, \Leftrightarrow\!\!\rightarrow, V \rangle$, where $s_1 \mid_f s_2 \in S$ whenever $s_1 \in S_1$ and $s_2 \in S_2$, $s_0 = s_{1,0} \mid_f s_{2,0}$, $\Leftrightarrow\!\!\rightarrow$ is inductively defined as follows:

- $s_1 \mid_f s_2 \overset{c}{\Leftrightarrow\!\!\rightarrow} s_1' \mid_f s_2'$ if $s_1 \overset{a}{\Leftrightarrow\!\!\rightarrow}_1 s_1'$, $s_2 \overset{b}{\Leftrightarrow\!\!\rightarrow}_2 s_2'$ and $f(a,b) = c$
- $s_1 \mid_f s_2 \overset{\epsilon(d)}{\Leftrightarrow\!\!\rightarrow} s_1' \mid_f s_2'$ if $s_1 \overset{\epsilon(d)}{\Leftrightarrow\!\!\rightarrow}_1 s_1'$ and $s_2 \overset{\epsilon(d)}{\Leftrightarrow\!\!\rightarrow}_2 s_2'$

and finally, the proposition assignment function V is defined by $V(s_1 \mid_f s_2) = V_1(s_1) \cup V_2(s_2)$.

Note also that the set of states and the transition relation of a timed transition system may be infinite. We shall use networks of timed automata as a finite syntactical representation to describe timed transition systems.

2.2 Networks of Timed Automata

A timed automaton [AD94, HNSY94] is a standard finite-state automaton extended with a finite collection of real-valued clocks. Conceptually, the clocks may be considered as the system clocks of a concurrent system. They are assumed to proceed at the same rate and measure the amount of time that has been elapsed since they were reset. The clocks values may be tested (compared with natural numbers) and reset (assigned to 0).

Definition 2.2 (Clock Constraints) *Let $\mathcal{C}$ be a set of real-valued clocks ranged over by x, y etc. We use $\mathcal{B}(\mathcal{C})$ to stand for the set of formulas ranged over by g, generated by the following syntax: $g ::= c \mid g \wedge g$, where c is an atomic constraint of the form: $x \sim n$ or $x \Leftrightarrow y \sim n$ for $x, y \in \mathcal{C}$, $\sim \in \{\leq, \geq, =, <, >\}$ and n being a natural number. We shall call $\mathcal{B}(\mathcal{C})$* clock constraints *or* clock constraint systems *over $\mathcal{C}$.* □

We shall use $\mathtt{tt}$ to stand for a constraint like $x \geq 0$ which is always true, and $\mathtt{ff}$ for a constraint $x < 0$ which is always false as clocks can only have non-negative values.

Definition 2.3 (Timed Automata) *A timed automaton A over actions $\mathcal{A}$, atomic propositions $\mathcal{P}$ and clocks $\mathcal{C}$ is a tuple $\langle N, l_0, \Leftrightarrow\!\!\rightarrow, I, V \rangle$. N is a finite set of nodes (control-nodes), l_0 is the initial node, and $\Leftrightarrow\!\!\rightarrow \subseteq N \times \mathcal{B}(\mathcal{C}) \times \mathcal{A} \times 2^{\mathcal{C}} \times N$ corresponds to the set of edges. In the case, $\langle l, g, a, r, l' \rangle \in \Leftrightarrow\!\!\rightarrow$ we shall write, $l \overset{g\,a\,r}{\Leftrightarrow\!\!\rightarrow} l'$ which represents*

[1] We extend the transition relation of a timed transition system such that $s \xrightarrow{0} s'$ iff $s = s'$.

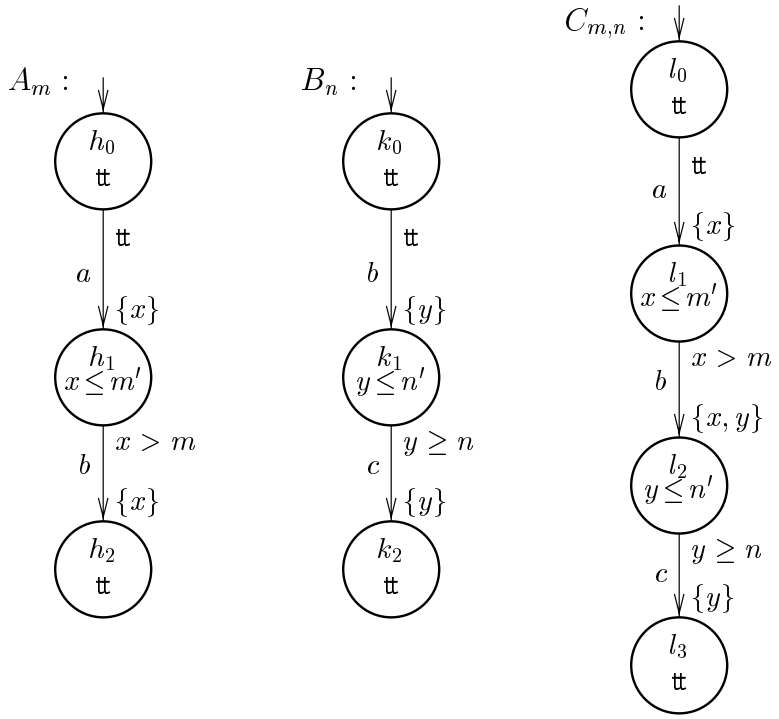


Figure 1: Three timed automata

an edge from the node l to the node l' with clock constraint g (also called the enabling condition of the edge), action a to be performed and the set of clocks r to be reset. $I : N \to \mathcal{B}(\mathcal{C})$ is a function which for each node assigns a clock constraint (also called the invariant condition of the node), and finally, $V : N \to 2^{\mathcal{P}}$ is a proposition assignment function which for each node gives a set of atomic propositions true in the node. □

Note that for each node l, there is an invariant condition $I(l)$ which is a clock constraint. Intuitively, this constraint must be satisfied by the system clocks whenever the system is operating in that particular control-node.

Informally, the system starts at node l_0 with all its clocks initialised to 0. The values of the clocks increase synchronously with time at node l as long as they satisfy the invariant condition $I(l)$. At any time, the automaton can change node by following an edge $l \overset{g\,a\,r}{\hookrightarrow} l'$ provided the current values of the clocks satisfy the enabling condition g. With this transition the clocks in r get reset to 0.

Example 2.1 *Consider the automata A_m, B_n and $C_{m,n}$ in Figure 1 where m, n, m', n' are natural numbers. We use m, n, m' and n' as parameters. The automaton $C_{m,n}$ has four nodes, l_0, l_1, l_2 and l_3, two clocks x and y, and three edges. The edge between l_1 and l_2 has b as action, $\{x, y\}$ as reset set and the enabling condition for the edge is $x > m$. The invariant conditions for nodes l_1 and l_2 are $x \leq m'$ and $y \leq n'$ respectively.*

Now we introduce the notion of a *clock assignment*. Formally, a clock assignment u for $\mathcal{C}$ is a function from $\mathcal{C}$ to $\mathbb{R}_+$. We denote by $\mathbb{R}_+^{\mathcal{C}}$ the set of clock assignments for

$\mathcal{C}$. For $u \in \mathbb{R}_+^{\mathcal{C}}$, $x \in \mathcal{C}$ and $d \in \mathbb{R}_+$, $u \oplus d$ denotes the time assignment which maps each clock x in $\mathcal{C}$ to the value $u(x) + d$. For $r \subseteq \mathcal{C}$, $r[u]$ denotes the assignment for $\mathcal{C}$ which maps each clock in r to the value 0 and agrees with u over $\mathcal{C} \backslash r$. Whenever $u \in \mathbb{R}_+^{\mathcal{C}}$, $v \in \mathbb{R}_+^K$ and $\mathcal{C}$ and K are disjoint, we use uv to denote the clock assignment over $\mathcal{C} \cup K$ such that $(uv)(x) = u(x)$ if $x \in \mathcal{C}$ and $(uv)(x) = v(x)$ if $x \in K$. Given a clock constraint $g \in \mathcal{B}(\mathcal{C})$ and a clock assignment $u \in \mathbb{R}_+^{\mathcal{C}}$, $g(u)$ is a boolean value describing whether g is satisfied by u or not. When $g(u)$ is true, we shall say that u is a solution to g.

A *state* of an automaton A is a pair (l, u) where l is a node of A and u a clock assignment for $\mathcal{C}$. The initial state of A is (l_0, u_0) where u_0 is the initial clock assignment mapping all clocks in $\mathcal{C}$ to 0. The semantics of A is given by the timed transition system $\mathcal{S}_A = \langle S, \sigma_0, \Leftrightarrow\!\!\rightarrow, V \rangle$, where S is the set of states of A, σ_0 is the initial state (l_0, u_0), $\Leftrightarrow\!\!\rightarrow$ is the transition relation defined as follows:

- $(l, u) \overset{a}{\Leftrightarrow\!\!\rightarrow} (l', u')$ if there exist r, g such that $l \overset{g\,a\,r}{\Leftrightarrow\!\!\rightarrow} l'$, $g(u)$ and $u' = r[u]$
- $(l, u) \overset{\epsilon(d)}{\Leftrightarrow\!\!\rightarrow} (l', u')$ if $l = l'$, $u' = u \oplus d$ and $I(u')$

and V is extended to S simply by $V(l, u) = V(l)$.

Example 2.2 *Reconsider the automaton $C_{m,n}$ of Figure 1. Assume that $d \geq 0$, $m < e \leq m'$ and $n \leq f \leq n'$. We have the following typical transition sequence:*

$$(l_0, (0,0)) \overset{\epsilon(d)}{\Leftrightarrow\!\!\rightarrow} (l_0, (d,d)) \overset{a}{\Leftrightarrow\!\!\rightarrow} (l_1, (0,d)) \overset{\epsilon(e)}{\Leftrightarrow\!\!\rightarrow} (l_1, (e, d+e)) \overset{b}{\Leftrightarrow\!\!\rightarrow}$$
$$(l_2, (0,0)) \overset{\epsilon(f)}{\Leftrightarrow\!\!\rightarrow} (l_2, (f,f)) \overset{c}{\Leftrightarrow\!\!\rightarrow} (l_3, (f,0))$$

Note that we need to assume that $m < e \leq m'$ and $n \leq f \leq n'$ because of the invariant conditions on l_1 and l_2.

Parallel composition may now be extended to timed automata in the obvious way: for two timed automata A and B and a synchronisation function f, the parallel composition $A \mid_f B$ denotes the timed transition system $\mathcal{S}_A \mid_f \mathcal{S}_B$. Note that the timed transition system $\mathcal{S}_A \mid_f \mathcal{S}_B$ can also be represented finitely as a timed automaton. In fact, one may effectively construct the product automaton $A \otimes_f B$ such that its timed transition system $\mathcal{S}_{A \otimes_f B}$ is bisimilar to $\mathcal{S}_A \mid_f \mathcal{S}_B$. The nodes of $A \otimes_f B$ is simply the product of A's and B's nodes, the invariant conditions on the nodes of $A \otimes_f B$ are the conjunctions of the conditions on respective A's and B's nodes, the set of clocks is the (disjoint) union of A's and B's clocks, and the edges are based on synchronisable A and B edges with enabling conditions conjuncted and reset-sets unioned.

Example 2.3 *Let f be the synchronisation function defined by $f(a, 0) = a$, $f(b, b) = b$ and $f(0, c) = c$. Then the automaton $C_{m,n}$ in Figure 1 is isomorphic to the part of $A_m \otimes_f B_n$ which is reachable from (h_0, k_0).*

3 A Logic for Safety and Bounded Liveness Properties

It has been pointed out, that the practical goal of verification of real-time systems, is to verify simple safety properties such as deadlock-freeness and mutual exclusion [Hal93]. In [YPD94], we have shown that such properties can be verified on-the-fly by simple reachability analysis which avoids to construct the whole state-space of systems.

3.1 Syntax and Semantics

We shall present a timed modal logic to specify safety properties. In fact, the logic can also be used to specify bounded liveness properties such as "whenever p becomes true, q will be true within a given time bound". The logic may be seen as a fragment of the timed μ-calculus presented in [HNSY94], and also studied in [LLW95].

Definition 3.1 *Let K be a finite set of clocks. We shall call K formula clocks. Let Id be a set of identifiers. The set $\mathcal{L}_s$ of formulae over K, Id, $\mathcal{A}$, and $\mathcal{P}$ is generated by the abstract syntax with φ and ψ ranging over $\mathcal{L}_s$:*

$$\varphi \quad ::= \quad cp \mid cp \vee \varphi \mid \varphi \wedge \psi \mid \forall\!\!\!\forall \varphi \mid [a]\,\varphi \mid z \,\mathsf{in}\, \varphi \mid Z$$

where cp may be an atomic clock constraint c in the form of $x \sim n$ or $x \Leftrightarrow y \sim n$ for $x, y \in K$ and natural number n, or an atomic proposition $p \in \mathcal{P}$, $a \in \mathcal{A}$ (an action), $z \in K$ and $Z \in \mathsf{Id}$ (an identifier). □

Note that the logic is essentially the fragment of the timed modal logic presented in [LLW95] by eliminating existential quantification over delay transitions, general disjunction over formulas, and existential quantification over a-transitions.

We do allow a simple form of disjunction, in that a clock constraint or an atomic proposition may be disjuncted with an arbitrary formula. We disallow general disjunction in the logic to achieve efficient compositional and symbolic model-checking algorithms. However, the logic is expressive enough to specify safety and bounded liveness properties. We shall see, that the simple form of disjunction allows us to specify bounded liveness properties such as "p will be true within n".

The meaning of the identifiers is specified by a declaration $\mathcal{D}$ assigning a formula of $\mathcal{L}_s$ to each identifier. When $\mathcal{D}$ is understood we write $Z \stackrel{def}{=} \varphi$ for $\mathcal{D}(Z) = \varphi$.

Given a timed transition system $\mathcal{S} = \langle S, s_0, \Leftrightarrow\!\!\rightarrow, V \rangle$ described by a network of timed automata, we interpret the $\mathcal{L}_s$ formulas over an extended state $\langle s, u \rangle$ where $s \in S$ is a state of $\mathcal{S}$, and u is a clock assignment for K. A formula of the form: $x \sim m$ and $x \Leftrightarrow y \sim n$ is satisfied by an extended state $\langle s, u \rangle$ if the values of x, y in u satisfy the required relationship. Informally, an extended state $\langle s, u \rangle$ satisfies $\forall\!\!\!\forall \varphi$ means that all future states reachable from $\langle s, u \rangle$ by delays will satisfy property φ; $\forall\!\!\!\forall$ denotes universal quantification over delay transitions. Similarly, a state $\langle s, u \rangle$ satisfies $[a]\varphi$ means that all intermediate states reachable from $\langle s, u \rangle$ by an a-transition (performed by s will satisfy property φ; $[a]$ denotes universal quantification over a-transitions.

$$
\begin{array}{rcl}
\langle s,u\rangle \models_{\mathcal{D}} c & \Rightarrow & c(u) \\
\langle s,u\rangle \models_{\mathcal{D}} p & \Rightarrow & p \in V(s) \\
\langle s,u\rangle \models_{\mathcal{D}} cp \vee \varphi & \Rightarrow & \langle s,u\rangle \models_{\mathcal{D}} cp \text{ or } \langle s,u\rangle \models_{\mathcal{D}} \varphi \\
\langle s,u\rangle \models_{\mathcal{D}} \varphi \wedge \psi & \Rightarrow & \langle s,u\rangle \models_{\mathcal{D}} \varphi \text{ and } \langle s,u\rangle \models_{\mathcal{D}} \psi \\
\langle s,u\rangle \models_{\mathcal{D}} \mathbb{V}\varphi & \Rightarrow & \forall d, s' : s \overset{\epsilon(d)}{\Leftrightarrow} s' \Rightarrow \langle s', u \oplus d\rangle \models_{\mathcal{D}} \varphi \\
\langle s,u\rangle \models_{\mathcal{D}} [a]\,\varphi & \Rightarrow & \forall s' : s \overset{a}{\Leftrightarrow} s' \Rightarrow \langle s', u\rangle \models_{\mathcal{D}} \varphi \\
\langle s,u\rangle \models_{\mathcal{D}} x \,\mathsf{in}\, \varphi & \Rightarrow & \langle s,u'\rangle \models_{\mathcal{D}} \varphi \text{ where } u' = \{x\}[u] \\
\langle s,u\rangle \models_{\mathcal{D}} Z & \Rightarrow & \langle s,u\rangle \models_{\mathcal{D}} \mathcal{D}(Z)
\end{array}
$$

Table 1: Definition of Satisfiability.

The formula $(x \,\mathsf{in}\, \varphi)$ initialises the formula clock x to 0; i.e. an extended state satisfies the formula in case the modified state with x being reset to 0 satisfies φ. Finally, an extended state satisfies an identifier Z if it satisfies the corresponding declaration (or definition) $\mathcal{D}(Z)$.

Let $\mathcal{D}$ be a declaration. Formally, the satisfaction relation $\models_{\mathcal{D}}$ between extended states and formulae is defined as the largest relation satisfying the implications of Table 1. Any relation satisfying the implications in Table 1 is called a *satisfiability* relation. It follows from standard fixpoint theory [Tar55] that $\models_{\mathcal{D}}$ is the union of all satisfiability relations. For simplicity, we shall omit the index $\mathcal{D}$ and write $\models$ instead of $\models_{\mathcal{D}}$ whenever it is understood from the context. We say that $\mathcal{S}$ satisfies a formula φ and write $\mathcal{S} \models \varphi$ when $\langle s_0, v_0\rangle \models \varphi$ where s_0 is the initial state of $\mathcal{S}$ and v_0 is the assignment with $v_0(x) = 0$ for all x. Similarly, we say that a timed automaton A satisfies φ in case $\mathcal{S}_A \models \varphi$. We write $A \models \varphi$ in this case.

Example 3.1 *Consider the following declaration $\mathcal{F}$ of the identifiers X_i and Z_i where i is a natural number.*

$$\mathcal{F} \;=\; \Big\{\; X_i \overset{def}{=} [a]\big(z \,\mathsf{in}\, Z_i\big), \quad Z_i \overset{def}{=} (\mathsf{at}(l_3) \vee \big(z < i \wedge [a]Z_i \wedge [b]Z_i \wedge [c]Z_i \wedge \mathbb{V}Z_i\big)\;\Big\}$$

Assume that $\mathsf{at}(l_3)$ is an atomic proposition meaning that the system is operating in control-node l_3. Then, X_i expresses the property that after an a-transition, the system must reach node l_3 within i time units.

Now, reconsider the automata A_m, B_n and $C_{m,n}$ of Figure 1 and Examples 2.1 and 2.2. Then it may be argued that $C_{m,n} \models X_{m'+n'}$ and (consequently), that $A_m \mid_f B_n \models X_{m'+n'}$.

3.2 Derived Operators

The property Z_i described in Example 3.1 is an attempt to specify a bounded liveness property: namely that a certain proposition must be satisfied within a given time bound. We shall use the more informative notation $\mathsf{at}(l_3)$ BEFORE i to denote Z_i. In the following, we shall present several such intuitive operators that are definable in our logic.

$$
\begin{array}{rcl}
\mathsf{INV}(\varphi) & \equiv & X \text{ where } X \stackrel{def}{=} \varphi \wedge \forall\!\!\forall X \wedge [\mathcal{A}]X \\
\varphi \ \mathsf{UNTIL}\ cp & \equiv & X \text{ where } X \stackrel{def}{=} cp \vee \Big(\varphi \wedge \forall\!\!\forall X \wedge [\mathcal{A}]X\Big) \\
\varphi \ \mathsf{UNTIL}_{<n}\ cp & \equiv & z \ \mathsf{in}\ \Big((\varphi \wedge (z < n))\ \mathsf{UNTIL}\ cp\Big) \\
cp \ \mathsf{BEFORE}\ n & \equiv & \mathrm{tt}\ \mathsf{UNTIL}_{<n}\ cp
\end{array}
$$

Table 2: Derived Operators.

For simplicity, we shall assume that the set of actions $\mathcal{A}$ is a finite set $\{a_1 ... a_m\}$, and use $[\mathcal{A}]\varphi$ to denote the formula $[a_1]\varphi \wedge ... \wedge [a_m]\varphi$. Now, let φ be a general formula, cp be an atomic clock constraint or an atomic proposition and n be a natural number. A collection of derived operators are given in Table 2.

The intuitive meanings of these operators are as follows: $\mathsf{INV}(\varphi)$ is satisfied by a timed automaton means that the automaton must enjoy the property φ now, and for all future time points, the reachable states should satisfy $\mathsf{INV}(\varphi)$ (i.e. X), and after any action transition, the reachable states should again satisfy $\mathsf{INV}(\varphi)$ (i.e. X): namely that φ is an invariant property of the automaton. φ UNTIL cp is satisfied by a timed automaton means that the automaton enjoys the property cp now, or otherwise all reachable states by action transitions and delay transitions should satisfy φ. This simply means that φ must hold at least before cp becomes true. The bounded version of the UNTIL-construct φ $\mathsf{UNTIL}_{<n}$ cp is similar to φ UNTIL cp except that cp must be true within n time units. A simpler version of this operator is cp BEFORE n meaning that property cp must be true within n time units. Alternatively, φ $\mathsf{UNTIL}_{<n}$ cp can be defined as z in X where $X \stackrel{def}{=} cp \vee \Big(\varphi \wedge (x < n) \wedge \forall\!\!\forall X \wedge [\mathcal{A}]X\Big)$.

4 Symbolic Model-Checking

We have presented a model to describe real-time systems, i.e. networks of timed automata, and a logic to specify properties of such systems. The next question is how to check whether a given formula in the logic is satisfied by a given network of automata. This is the so-called model-checking problem. As the systems we are studying are in general infinite-state due to the real-valued clocks, we need efficient methods to represent the state-space symbolically. The region-graph technique by Alur, Courcoubetis and Dill allows the state space of a real time system to be partitioned into finitely many regions in such a way that states within the same region satisfy the same properties [ACD90, AD94]. It follows that model-checking is decidable as the region partitioning enables standard finite-state algorithmic model-checking techniques to be applied. However, as the notion of region is property-independent and the number of such regions depends on the constants used in the clock constraints of an automaton, this leads to an extremely fine (and large) partitioning.

Recall that a semantical state of a network of timed automata is a pair (l, u) where l is a control-node and $u \in \mathbb{R}_+^{\mathcal{C}}$ is a clock assignment. The model-checking problem

is in general to check whether an *extended state* in the form $\langle (l,u),v\rangle$ satisfies a given formula φ, that is,

$$\langle (l,u),v\rangle \models \varphi$$

Note that u is a clock assignment for the automata clocks and v is a clock assignment for the formula clocks. Now, the problem is that we have infinitely many such assignments to check in order to conclude $\langle (l,u),v\rangle \models \varphi$.

In this section, we shall use clock constraints $\mathcal{B}(\mathcal{C}\cup K)$ for automata clocks $\mathcal{C}$ and formula clocks K, as defined in Section 2 to symbolically represent clock assignments. We shall use D to range over $\mathcal{B}(\mathcal{C}\cup K)$. Instead of checking $\langle (l,u),v\rangle \models \varphi$ for each u and v, we develop an algorithm to simultaneously check

$$(l,D)\vdash \varphi$$

which means that for each u and v such that uv is a solution to the constraint system D, we have $\langle (l,u),v\rangle \models \varphi$.

Thus the space $\mathbb{R}_+^{\mathcal{C}\cup K}$ is partitioned in terms of clock constraints. As the partitioning takes into account both the network and the particular property to check, the number of partitions is in practice considerably smaller compared with the region-technique.

4.1 Operations on Clock Constraints

To develop the model-checking algorithm, we need a few operations to manipulate clock constraints. Given a clock constraint D, we call the set of clock assignments satisfying D, the *solution set* of D.

Definition 4.1 *Let A and A' be the solution sets of clock constraints $D, D' \in \mathcal{B}(\mathcal{C}\cup K)$. We define*

$$\begin{array}{rcl} A^{\uparrow} & = & \{\ w+d \mid w\in A \text{ and } d\in \mathbb{R}_+\ \} \\ A^{\downarrow} & = & \{\ w \mid \exists d\in \mathbb{R}_+ :\ w+d\in A\ \} \\ \{x\}A & = & \{\ x[w] \mid w\in A\ \} \\ A\wedge A' & = & \{\ w \mid w\in A \text{ and } w\in A'\ \} \end{array}$$

□

First, note that $A\wedge A'$ is simply the intersection of the two sets. Intuitively, $A^{\downarrow}$ is the largest set of time assignments that will eventually reach A after some delay; whereas $A^{\uparrow}$ is the dual of $A^{\downarrow}$: namely that it is the largest set of time assignments that can be reached by some delay from A. Finally, $\{x\}A$ the set of time assignments A where x is reset to zero. We extend the reset operator to sets of clocks. Let $r=\{x_1,\ldots,x_n\}$ be a set of clocks. We define $r(A)$ recursively by $\{\}(A)=A$ and $\{x_1,\ldots,x_n\}(A)=\{x_1\}(\{x_2,\ldots,x_n\}A)$.

The following Proposition establishes that the class of clock constraints $\mathcal{B}(\mathcal{C}\cup K)$ is closed under the four operations defined above.

Proposition 4.1 (Closure Property of Clock Constraints) *Let $D, D' \in \mathcal{B}(\mathcal{C}\cup K)$ with solution sets A and A', and $x\in\mathcal{C}\cup K$. Then there exist $D_1, D_2, D_3, D_4 \in \mathcal{B}(\mathcal{C}\cup K)$ with solution sets $A^{\uparrow}$, $A^{\downarrow}$, $\{x\}A$ and $A\wedge A'$ respectively.*

$$
\begin{array}{lcl}
D = \emptyset & \Rightarrow & (l, D) \vdash_{\mathcal{D}} \varphi \\
(l, D) \vdash_{\mathcal{D}} c & \Rightarrow & D \subseteq c \\
(l, D) \vdash_{\mathcal{D}} p & \Rightarrow & p \in V(l) \\
(l, D) \vdash_{\mathcal{D}} c \vee \varphi & \Rightarrow & (l, D \wedge \neg c) \vdash_{\mathcal{D}} \varphi \\
(l, D) \vdash_{\mathcal{D}} p \vee \varphi & \Rightarrow & (l, D) \vdash_{\mathcal{D}} p \text{ or } (l, D) \vdash_{\mathcal{D}} \varphi \\
(l, D) \vdash_{\mathcal{D}} \varphi \wedge \psi & \Rightarrow & (l, D) \vdash_{\mathcal{D}} \varphi \text{ and } (l, D) \vdash_{\mathcal{D}} \psi \\
(l, D) \vdash_{\mathcal{D}} \Box\varphi & \Rightarrow & (l, D) \vdash_{\mathcal{D}} \varphi \text{ and } (l, (D \wedge I(l))^{\uparrow} \wedge I(l)) \vdash_{\mathcal{D}} \varphi \\
(l, D) \vdash_{\mathcal{D}} [a]\,\varphi & \Rightarrow & (l', r(D \wedge g)) \vdash_{\mathcal{D}} \varphi \text{ whenever } l \overset{g\,a\,r}{\Longrightarrow} l' \\
(l, D) \vdash_{\mathcal{D}} x \text{ in } \varphi & \Rightarrow & (l, \{x\}D) \vdash_{\mathcal{D}} \varphi \\
(l, D) \vdash_{\mathcal{D}} Z & \Rightarrow & (l, D) \vdash_{\mathcal{D}} \mathcal{D}(Z)
\end{array}
$$

Table 3: Definition of Symbolic Satisfiability.

PROOF: See [Dil89, BL96]. □

In fact, the resulted constraints D_i's can be effectively constructed from D and D', as shown in Section 4.3. In order to save notation, we shall use $D^{\uparrow}$, $D^{\downarrow}$, $\{x\}D$ and $D \wedge D'$ to denote the clock constraints which must exist due to the above proposition. We also need a few *predicates* over clock constraints for the model-checking procedure. We write $D \subseteq D'$ to mean that the solution set of D is included in the solution set of D' and $D = \emptyset$ to mean that the solution set of D is empty.

4.2 Model-Checking by Constraint Solving

Given a network of timed automaton A over clocks $\mathcal{C}$, we shall interpret formulas over clocks K with respect to symbolic states of the form (l, D) where l is a control-node of A and D is a clock constraint of $\mathcal{B}(\mathcal{C} \cup K)$.

Let $\mathcal{D}$ be a declaration. The symbolic satisfaction relation $\vdash_{\mathcal{D}}$ between symbolic states and formulas is defined as the largest relation satisfying the implications in Table 3. We call a relation satisfying the implications in Table 3 a *symbolic satisfiability* relation. Again, it follows from standard fixpoint theory [Tar55] that $\vdash_{\mathcal{D}}$ is the union of all symbolic satisfiability relations. For simplicity, we shall omit the index $\mathcal{D}$ and write $\vdash_{\mathcal{D}}$ instead of $\vdash$ whenever it is understood from the context.

The following Theorem shows that the symbolic interpretation of $\mathcal{L}_s$ in Table 3 expresses the sufficient and necessary conditions for a timed automata to satisfy a formula φ[2].

Theorem 4.1 (Correctness of Symbolic Satisfiability) *Let A be a timed automaton over clock set $\mathcal{C}$ and φ a formula over K. Then the following holds:*

$$A \models \varphi \quad \textit{if and only if} \quad (l_0, D_0) \vdash \varphi$$

[2]Note that Theorem cannot be extended to a logic with general disjunction (or existential quantifications): the obvious requirement that $(l, D) \vdash \varphi_1 \vee \varphi_2$ should imply either $(l, D) \vdash \varphi_1$ or $(l, D) \vdash \varphi_2$ will fail to satisfy the Theorem.

where l_0 is the initial node of A and D_0 is the linear constraint system $\{x = 0 \mid x \in \mathcal{C} \cup K\}$.

PROOF: We prove the theorem by structural induction over φ. As induction hypothesis (I.H.) assume that $\Big(\forall uv \in D : \langle (l,u), v\rangle \models \varphi \iff (l, D) \vdash \varphi \Big)$ for all symbolic states (l, D). We have one case for each operator in the logic:

$$
\begin{array}{rcll}
(l,D) \vdash c & \iff & D \subseteq c & \text{(by Tab. 3)} \\
& \iff & c(v) & \text{(by Def. 4.1)} \\
& \iff & \forall uv \in D : \langle (l,u), v\rangle \models c & \text{(by Tab. 1)} \\
(l,D) \vdash p & \iff & p \in V(l) & \text{(by Tab. 3)} \\
& \iff & \forall uv \in D : \langle (l,u), v\rangle \models p & \text{(by Tab. 1)} \\
(l,D) \vdash c \vee \varphi & \iff & (l, D \wedge \neg c) \vdash \varphi & \text{(by Tab. 3)} \\
& \iff & \forall uv \in (D \wedge \neg c) : \langle (l,u), v\rangle \models \varphi & \text{(by I.H.)} \\
& \iff & \forall uv \in D : \Big(\neg c(v) \Rightarrow \langle (l,u), v\rangle \models \varphi\Big) & \\
& \iff & \forall uv \in D : \langle (l,u), v\rangle \models c \ \vee \varphi & \text{(by Tab. 1)} \\
(l,D) \vdash p \vee \varphi & \iff & (l,D) \vdash p \text{ or } (l,D) \vdash \varphi & \text{(by Tab. 3)} \\
& \iff & p \in V(l) \text{ or } \forall uv \in D : \langle (l,u), v\rangle \models \varphi & \text{(by I.H.)} \\
& \iff & \forall uv \in D : \langle (l,u), v\rangle \models p \vee \varphi & \text{(by Tab. 1)} \\
(l,D) \vdash \varphi \wedge \psi & \iff & (l,D) \vdash \varphi \text{ and } (l,D) \vdash \psi & \text{(by Tab. 3)} \\
& \iff & \forall uv \in D : \langle (l,u), v\rangle \models \varphi \text{ and} & \text{(by I.H.)} \\
& & \forall uv \in D : \langle (l,u), v\rangle \models \psi & \\
& \iff & \forall uv \in D : \langle (l,u), v\rangle \models \varphi \wedge \psi & \text{(by Tab. 1)} \\
(l,D) \vdash [a]\varphi & \iff & (l', r(D \wedge g)) \vdash \varphi \text{ whenever } l \overset{g\,a\,r}{\Longrightarrow} l' & \text{(by Tab. 3)} \\
& \iff & \forall uv \in r(D \wedge g) : \langle (l',u), v\rangle \models \varphi & \text{(by I.H.)} \\
& & \text{whenever } l \overset{g\,a\,r}{\Longrightarrow} l' & \\
& \iff & \forall uv \in D : \langle (l, r[u]), v\rangle \models \varphi & \text{(by Def. 4.1)} \\
& & \text{whenever } l \overset{g\,a\,r}{\Longrightarrow} l' \text{ and } g(u) & \\
& \iff & \forall uv \in D : \langle (l,u), v\rangle \models [a]\varphi & \text{(by Tab. 1)} \\
(l,D) \vdash \mathbb{W}\varphi & \iff & (l,D) \vdash \varphi \text{ and } (l, (D \wedge I(l))^{\uparrow} \wedge I(l)) \vdash \varphi & \text{(by Tab. 3)} \\
& \iff & \forall uv \in D : \langle (l,u), v\rangle \models \varphi \text{ and} & \text{(by I.H.)} \\
& & \forall uv \in ((D \wedge I(l))^{\uparrow} \wedge I(l)) : \langle (l,u), v\rangle \models \varphi & \\
& \iff & \forall uv \in D, d \in \mathbb{R}_+ : \langle (l, u \oplus d), v \oplus d\rangle \models \varphi & \text{(by Def. 4.1)} \\
& & \text{whenever } (l,u) \overset{\epsilon(d)}{\Longrightarrow} (l, u \oplus d) \text{ and } I(l)(u \oplus d) & \\
& \iff & \forall uv \in D : \langle (l,u), v\rangle \models \mathbb{W}\varphi & \text{(by Tab. 1)} \\
(l,D) \vdash x \text{ in } \varphi & \iff & (l, \{x\}D) \vdash \varphi & \text{(by Tab. 3)} \\
& \iff & \forall uv \in \{x\}D : \langle (l,u), v\rangle \models \varphi & \text{(by I.H.)} \\
& \iff & \forall uv \in D : \langle (l, x[u]), v\rangle \models \varphi & \text{(by Def. 4.1)} \\
& \iff & \forall uv \in D : \langle (l,u), v\rangle \models x \text{ in } \varphi & \text{(by Tab. 1)}
\end{array}
$$

$$\begin{array}{lll} (l,D) \vdash Z & \Longleftrightarrow & (l,D) \vdash \mathcal{D}(Z) \qquad \text{(by Tab. 3)} \\ & \Longleftrightarrow & \forall uv \in D : \langle (l,u), v\rangle \models \mathcal{D}(Z) \qquad \text{(by I.H.)} \\ & \Longleftrightarrow & \forall uv \in D : \langle (l,u), v\rangle \models Z \qquad \text{(by Tab. 1)} \end{array}$$

□

Given a symbolic satisfaction problem $(l, D) \vdash \varphi$ we may determine its validity by using the implications of Table 3 as rewrite rules. Due to the maximal fixed point property of $\vdash$, rewriting may be terminated successfully in case cycles are encountered.

Example 4.1 *Reconsider the automaton $C_{m,n}$ in Figure 1 assuming that $m' = n' = +\infty$ (making the invariants of l_1 and l_2 true). Consider the property $[a](z \,\mathsf{in}\, X)$ where X is defined as follows:*

$$X \stackrel{\text{def}}{=} (z \geq i) \vee ([c]\mathsf{ff} \wedge [a]X \wedge [b]X \wedge \forall X)$$

The property $[a](z \,\mathsf{in}\, X)$ expresses that the accumulated time between an initial a-action and a following c-action must exceed i. We want to show that $C_{m,n}$ satisfies this property provided the sum of the delays m and n exceeds the required delay i. That is, we must show $(l_0, D_0) \vdash [a](z \,\mathsf{in}\, X)$ provided $n + m \geq i$. The generated rewrite tree (i.e. execution tree of our model-checking procedure) is illustrated in Figure 2. In the rewrite tree, a node (i.e. a problem) is related to its sons by application of the appropriate rewrite rule of Table 3: i.e. the sons represent the conjuncts of the right-hand side of the applied rule[3]. The leaves of the tree are either obviously valid problems or re-occurrences. The leaf-problem labeled (c) is valid as $(D_3 \wedge y \geq n) = \emptyset$ holds under the assumption that $n + m \geq i$. Thus (c) is an instance of the first rule of Table 3. The problem labeled (b) is a re-occurrence of the earlier problem $(l_1, {D_0}^{\uparrow})$ as it can be shown that ${D_0}^{\uparrow} = {D_1}^{\uparrow}$.

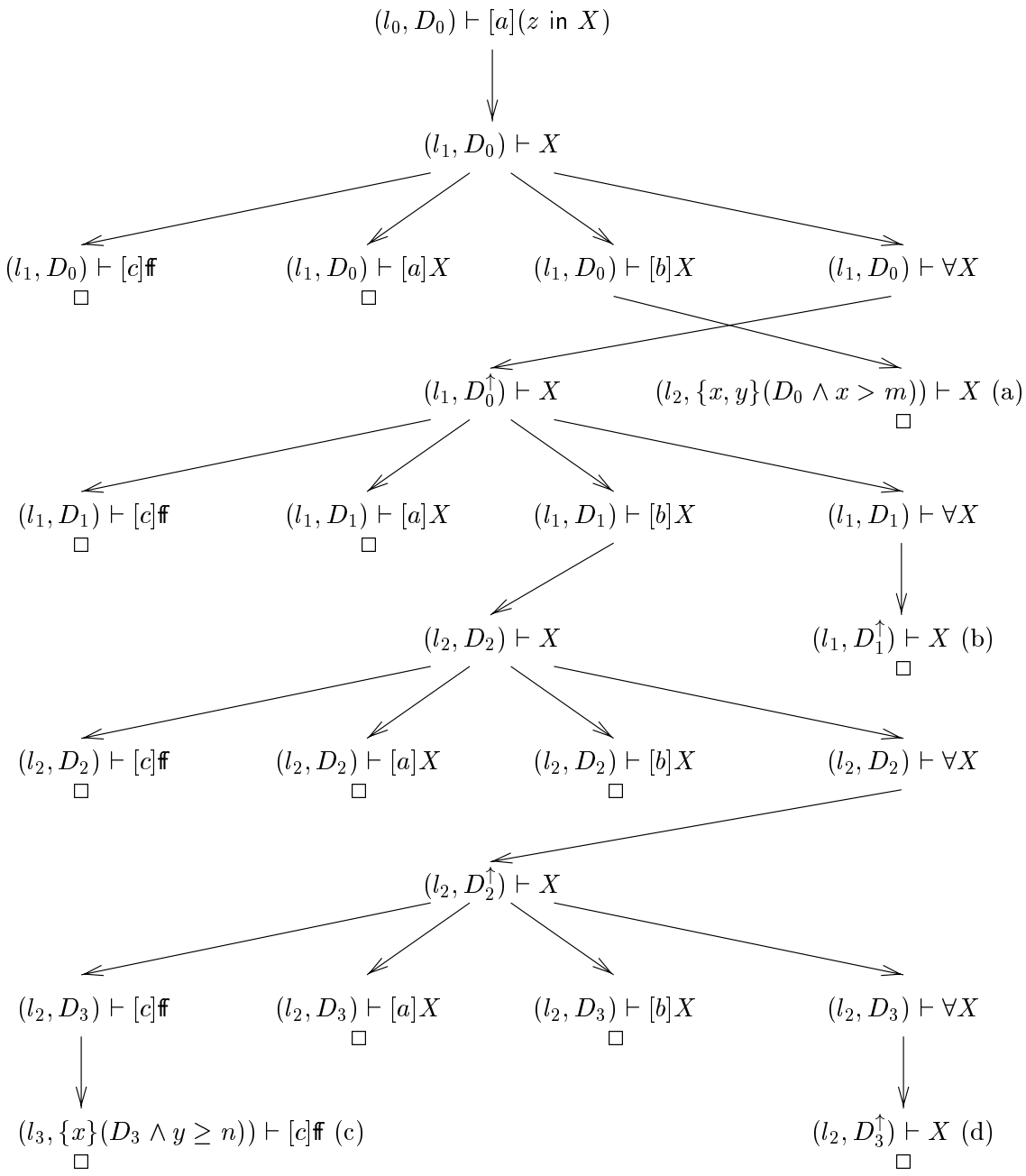
4.3 Implementation Issues

The operations and predicates on clock constraint systems discussed in Section 4.1 can be efficiently implemented by representing constraint systems as weighted directed graphs [Bel57, Dil89, BL96]. The basic idea is to use a shortest-path algorithm to close a constraint system under entailment so that operations and predicates can be easily computed.

Given a clock constraint system D over a clock set $\mathcal{C}$, we represent D as a weighted directed graph with vertices $\mathcal{C} \cup \{0\}$. The graph will have an edge from x to y with weight m provided $x \Leftrightarrow y \leq m$ is a constraint of D. Similarly, there will be an edge from 0 to x (from x to 0) with weight m whenever $x \leq m$ ($x \geq \Leftrightarrow m$) is a constraint of D[4].

[3] For problems involving an identifier, the tree reflects *two* successive rule applications starting with the unfolding of the identifier.

[4] In this presentation we have made the simplifying assumption that D does not contain any strict constraints, i.e. constraints of the form $x - y < n$.



$D_0 = \{x = y = z = 0\}$ $\quad D_0^{\uparrow} = \{x = y = z\}$

$D_1 = D_0^{\uparrow} \wedge (z < i) \equiv \{x = y = z, z < i\}$ $\quad D_1^{\uparrow} = D_0^{\uparrow}$

$D_2 = \{x, y\}(D_1 \wedge (x > m)) \equiv \{x = y = 0, m < z < i\}$ $\quad D_2^{\uparrow} = \{x = y, m < z \Leftrightarrow x < i\}$

$D_3 = D_2^{\uparrow} \wedge z < i = \{x = y, m < z \Leftrightarrow x < i, z < i\}$ $\quad D_3^{\uparrow} = D_2^{\uparrow}$

(a) $(D_0 \wedge (x > m)) = \emptyset$. $\quad$ (b) $D_1^{\uparrow} = D_0^{\uparrow}$, i.e. re-occurrence.

(c) $(D_3 \wedge (y \geq n)) = \emptyset$, if $(n + m) \geq i$. $\quad$ (d) $D_3^{\uparrow} = D_2^{\uparrow}$, i.e. re-occurrence.

Figure 2: Rewrite Tree of $(l_0, D_0) \vdash [a](z \text{ in } X)$.

A clock constraint system D is *closed under entailment* if no constraint of D can be strengthened without reducing the solution set. For closed constraint systems D and D' the inclusion and emptiness predicates are easy to decide: $D \subseteq D'$ holds iff for all constraint in D' there is a tighter constraint in D (e.g. whenever $(x \Leftrightarrow y \leq m) \in D'$ then $(x \Leftrightarrow y \leq m') \in D$ for some $m' \leq m$); $D = \emptyset$ holds if D contains two contradicting constraints (e.g. $x \Leftrightarrow y \leq m$ and $x \Leftrightarrow y \geq n$ where $m < n$). To close a clock constraint system D amounts to solve the shortest-path problem for its graph and can thus be computed in $\mathcal{O}(n^3)$ (which is also the complexity for the inclusion and emptiness predicates), where n is the number of clocks.

Given constraint systems D and D' the operations $D^{\uparrow}$, $D^{\downarrow}$, $\{x\}D$ and $D \wedge D'$ can be computed in $\mathcal{O}(n^2)$. The complexity of the operation $c \wedge D$, where c is an atomic constraint, is $\mathcal{O}(1)$. The two predicates $D \subseteq D'$ and $D = \emptyset$ can be checked in $\mathcal{O}(n^2)$ and $\mathcal{O}(1)$ respectively.

5 Compositional Model-Checking

The symbolic model-checking presented in the previous section provides an efficient way to deal with the potential explosion caused by the addition of clocks. However, a potential explosion in the node-space due to parallel composition still remains. In this section we attack this problem by development of a quotient construction, which allows components to be gradually moved from the parallel system into the specification, thus avoiding explicit construction of the global node space. The intermediate specifications are kept small using minimisation heuristics. Recent experimental work by Andersen [And95] demonstrates that for (untimed) finite-state systems the quotient technique improves results obtained using Binary Decision Diagrams. Also, an initial experimental investigation of the quotient technique to real-time systems in [LL95] has indicated that these promising results will carry over to the setting of real-time systems. In this section we shall provide a new (and compared with [LL95] simple) quotient construction and show how to integrate it with the symbolic technique of the previous section.

5.1 Quotient Construction

Given a formula φ, and two timed automata A and B we aim at constructing a formula (called the *quotient*) $\varphi /_f B$ such that

$$A \mid_f B \models \varphi \quad \text{if and only if} \quad A \models \varphi /_f B$$

The bi-implication indicates that we are moving parts of the parallel system into the formula. Clearly, if the quotient is not much larger than the original formula, we have simplified the task of model-checking, as the (symbolic) semantics of A is significantly smaller than that of $A \mid_f B$. More precisely, whenever φ is a formula over K, B is a timed automaton over $\mathcal{C}$ and l is a node of B, we define the quotient formula $\varphi /_f l$

$$
\begin{array}{rcl}
c/_f\, l & = & c \\
p/_f\, l & = & \begin{cases} \mathsf{tt} & ; p \in V(l) \\ p & ; p \notin V(l) \end{cases} \\
(c \vee \varphi)/_f\, l & = & (c/_f\, l) \vee (\varphi/_f\, l) \\
(p \vee \varphi)/_f\, l & = & (p/_f\, l) \vee (\varphi/_f\, l) \\
(\varphi_1 \wedge \varphi_2)/_f\, l & = & (\varphi_1/_f\, l) \wedge (\varphi_2/_f\, l) \\
(\mathbb{V}\varphi)/_f\, l & = & \mathbb{V}\Big(I(l) \Rightarrow (\varphi/_f\, l)\Big) \\
([a]\varphi)/_f\, l & = & \displaystyle\bigwedge_{l \xrightarrow{g,c,r} l' \,\wedge\, f(b,c)=a} \Big(g \Rightarrow [b](r \text{ in } \varphi/_f\, l')\Big) \\
(x \text{ in } \varphi)/_f\, l & = & x \text{ in } (\varphi/_f\, l) \\
X/_f\, l & = & X_l \text{ where } X_l \stackrel{def}{=} \mathcal{D}(X)/_f\, l
\end{array}
$$

Table 4: Definition of Quotient $\varphi/_f\, l$

over $\mathcal{C} \cup K$ in Table 4 on the structure of φ[5][6].

The quotient $\varphi/_f\, l$ expresses the sufficient and necessary requirement to a timed automaton A in order that the parallel composition $A \mid_f B$ with B at node l satisfies φ. In most cases quotienting simply distributes with respect to the formula construction. The quotient construction for $\mathbb{V}\varphi$ reflects that $A \mid_f B$ can only delay provided $I(l)$ is satisfied. The quotient construction for $[a]\varphi$ must quantify over all actions of A which can possibly lead to an a-transition of $A \mid_f B$: according to the semantics of parallel composition, b is such an action provided B (at node l) can perform a synchronisable action c (according to some edge $l \overset{g\,c\,r}{\Leftrightarrow} l'$) such that $f(b,c) = a$. The guard as well as the reset set of the involved A-edge $l \overset{g\,c\,r}{\Leftrightarrow} l'$ is reflected in the quotient formula.

Note that the quotient construction for identifiers introduces new identifiers of the form X_l. These new identifiers and their definitions ($X_l \stackrel{def}{=} \mathcal{D}(X)/_f\, l$) are collected in the (quotient) declaration $\mathcal{D}_B$.

For l_0 the initial node of a timed automaton B, the quotient $\varphi/_f\, l_0$ expresses the sufficient and necessary requirement to a timed automaton A in order that the parallel composition $A \mid_f B$ satisfies φ. This is stated in the following Theorem 5.1:

Theorem 5.1 (Correctness of Quotienting) *Let A and B be two timed automata and let l_0 be the initial node of B. Then*

$$A \mid_f B \models_{\mathcal{D}} \varphi \qquad \text{if and only if} \qquad A \models_{\mathcal{D}_B} \left(\varphi/_f\, l_0\right)$$

[5] For $g = c_1 \wedge \ldots c_n$ a clock constraint we write $g \Rightarrow \varphi$ as an abbreviation for the formula $\neg c_1 \vee \ldots \vee \neg c_n \vee \varphi$. This is an $\mathcal{L}_s$-formula as atomic constraint are closed under negation.

[6] In the rule for $[a]\varphi$, we assume that all nodes l of a timed automaton are extended with a 0-edge $l \xrightarrow{\mathsf{tt},0,\emptyset} l$.

PROOF: By structural induction over φ. As induction hypothesis (I.H) assume $\Big(\langle s, uv\rangle \models_{\mathcal{D}_B} \varphi/_f\, l \iff \langle s|_f(l,u), v\rangle \models_{\mathcal{D}} \varphi\Big)$ for all clock assignments u, v and states $\langle l, u\rangle$ and s. We have one case for each operator in the logic (we omit the proof of $p \vee \varphi$ which is similar to the proof of $c \vee \varphi$):

$$
\begin{array}{rcll}
\langle s, uv\rangle \models c/_f\, l & \iff & \langle s|_f(l,u), v\rangle \models c & \text{(by Tab. 4)} \\
\langle s, uv\rangle \models p/_f\, l & \iff & \langle s|_f(l,u), v\rangle \models p & \text{(by Tab. 4)} \\
\langle s, uv\rangle \models (c \vee \varphi)/_f\, l & \iff & \langle s, uv\rangle \models (c/_f\, l) \vee (\varphi/_f\, l) & \text{(by Tab. 4)} \\
 & \iff & \langle s, uv\rangle \models c \text{ or } \langle s, uv\rangle \models \varphi & \text{(by Tab. 1)} \\
 & \iff & \langle s|_f(l,u), v\rangle \models c \text{ or } \langle s|_f(l,u), v\rangle \models \varphi & \text{(by I.H.)} \\
 & \iff & \langle s|_f(l,u), v\rangle \models c \vee \varphi & \text{(by Tab. 1)} \\
\langle s, uv\rangle \models (\varphi \wedge \psi)/_f\, l & \iff & \langle s, uv\rangle \models (\varphi/_f\, l) \wedge (\psi/_f\, l) & \text{(by Tab. 4)} \\
 & \iff & \langle s, uv\rangle \models (\varphi/_f\, l) \text{ and } \langle s, uv\rangle \models (\psi/_f\, l) & \text{(by Tab. 1)} \\
 & \iff & \langle s|_f(l,u), v\rangle \models \varphi \text{ and } \langle s|_f(l,u), v\rangle \models \psi & \text{(by I.H.)} \\
 & \iff & \langle s|_f(l,u), v\rangle \models \varphi \wedge \psi & \text{(by Tab. 1)} \\
\langle s, uv\rangle \models ([a]\varphi)/_f\, l & \iff & \langle s, uv\rangle \models \bigwedge_{l \xrightarrow{g\,c\,r} l' \wedge f(b,c)=a} (g \Rightarrow [b](r \text{ in } \varphi/_f\, l') & \text{(by Tab. 4)} \\
 & \iff & \langle s, uv\rangle \models g \Rightarrow [b](r \text{ in } \varphi/l') & \text{(by Tab. 1)} \\
 & & \text{whenever } l \overset{g\,c\,r}{\Leftrightarrow} l' \text{ and } f(b,c) = a & \\
 & \iff & \langle s', r[u]v\rangle \models \varphi/_f\, l' \text{ whenever } l \overset{g\,c\,r}{\Leftrightarrow} l', & \text{(by Tab. 1)} \\
 & & g(u),\ s \overset{a}{\Leftrightarrow} s' \text{ and } f(b,c) = a & \\
 & \iff & \langle s'|_f(l, r[u]), v\rangle \models \varphi/_f\, l' \text{ whenever} & \text{(by I.H.)} \\
 & & l \overset{g\,c\,r}{\Leftrightarrow} l', g(u),\ s \overset{a}{\Leftrightarrow} s' \text{ and } f(b,c) = a & \\
 & \iff & \langle s|_f(l,u)v\rangle \models [a]\varphi & \text{(by Tab. 1)} \\
\langle s, uv\rangle \models (\mathbb{V}\varphi)/_f\, l & \iff & \langle s, uv\rangle \models \mathbb{V}(I(l) \Rightarrow (\varphi/_f\, l)) & \text{(by Tab. 4)} \\
 & \iff & \forall d \in \mathbb{R}_+ : \langle s^d, (uv) \oplus d\rangle \models \varphi/_f\, l & \text{(by Tab. 1)} \\
 & & \text{whenever } s \overset{\epsilon(d)}{\Leftrightarrow} s' \text{ and } I(l)(u \oplus d) & \\
 & \iff & \forall d \in \mathbb{R}_+ : \langle s^d|_f(l, u \oplus d), v \oplus d\rangle \models \varphi & \text{(by I.H.)} \\
 & & \text{whenever } s \overset{\epsilon(d)}{\Leftrightarrow} s' \text{ and } I(l)(u \oplus d) & \\
 & \iff & \langle s|_f(l,u), v\rangle \models \varphi & \text{(by Tab. 1)} \\
\langle s, uv\rangle \models (x \text{ in } \varphi)/_f\, l & \iff & \langle s, uv\rangle \models x \text{ in } (\varphi/_f\, l) & \text{(by Tab. 4)} \\
 & \iff & \langle s, r[u]v\rangle \models \varphi/_f\, l & \text{(by Tab. 1)} \\
 & \iff & \langle s|_f(l, r[u]), v\rangle \models \varphi & \text{(by I.H.)} \\
 & \iff & \langle s|_f(l,u), v\rangle \models x \text{ in } \varphi & \text{(by Tab. 1)} \\
\langle s, uv\rangle \models_{\mathcal{D}_B} Z/_f\, l & \iff & \langle s, uv\rangle \models_{\mathcal{D}_B} Z_l & \text{(by Tab. 4)} \\
 & & \text{where } \mathcal{D}_B(Z_l) = \mathcal{D}(Z)/_f\, l & \\
 & \iff & \langle s, uv\rangle \models_{\mathcal{D}_B} \mathcal{D}(Z)/_f\, l & \text{(by Tab. 1)} \\
 & \iff & \langle s|_f(l,u), v\rangle \models_{\mathcal{D}} \mathcal{D}(Z) & \text{(by I.H.)} \\
 & \iff & \langle s|_f(l,u), v\rangle \models_{\mathcal{D}} Z & \text{(by Tab. 1)}
\end{array}
$$

□

Example 5.1 *Reconsider the network, synchronisation function and property from Examples 2.1, 2.2, 2.3 and 4.1. We want to establish that the network* $A_n \mid_f B_n$ *satisfies the following property* Y *provided* $n + m \geq i$*:*

$$\begin{aligned} Y &\stackrel{def}{=} [a]\big(z \text{ in } X\big) \\ X &\stackrel{def}{=} (z \geq i) \vee \big([c]\mathtt{ff} \wedge [a]X \wedge [b]X \wedge \mathbb{W}X\big) \end{aligned}$$

From Theorem 5.1 it follows that the sufficient and necessary requirement to A_n *in order that* $A_n \mid_f B_n$ *satisfies* Y *is that* A_n *satisfies* $Y /_f k_0$*. Using the quotient definition from Table 4 we get:*

$$\begin{aligned} Y /_f k_0 &\stackrel{def}{=} z \text{ in } (X /_f k_0) \\ X /_f k_0 &\stackrel{def}{=} (z \geq i) \vee \big([b](y \text{ in } X /_f k_1) \wedge \mathbb{W}(X /_f k_0)\big) \\ X /_f k_1 &\stackrel{def}{=} (z \geq i) \vee \big((y \geq n \Rightarrow [c]\mathtt{ff}) \wedge \mathbb{W}(X /_f k_1)\big) \end{aligned}$$

5.2 Minimisations

It is obvious that repeated quotienting leads to an explosion in the formula. The crucial observation made by Andersen in the (untimed) finite-state case is that simple and effective transformations of the formulas in practice may lead to significant reductions.

In presence of real-time we need, in addition to the minimisation strategies of Andersen, heuristics for propagating and eliminating constraints on clocks in formulas and declarations. Below we describe the transformations considered:

Reachability: When considering an initial quotient formula $\varphi /_f l_0$ not all identifiers in $\mathcal{D}_B$ may be reachable. Standard "on-the-fly" techniques can be applied to ensure that only the reachable part of $\mathcal{D}_B$ is generated.

Boolean Simplification: Formulas may be simplified using the following simple boolean equations and their duals: $\mathtt{ff} \wedge \varphi \equiv \mathtt{ff}$, $\mathtt{tt} \wedge \varphi \equiv \varphi$, $[a]\mathtt{tt} \equiv \mathtt{tt}$, $\mathbb{W}\mathtt{tt} \equiv \mathtt{tt}$, and $x \text{ in } \mathtt{ff} \equiv \mathtt{ff}$.

Constraint Propagation: Constraints on formula clocks may be propagated using various distribution laws (see Table 5). In some cases, propagation will lead to trivial clock constraints, which may be simplified to either $\mathtt{tt}$ or $\mathtt{ff}$ and hence made applicable to Boolean Simplification.

Constant Propagation: Identifiers with identifier-free definitions (i.e. constants such as $\mathtt{tt}$ or $\mathtt{ff}$) may be removed while substituting their definitions in the declaration of all other identifiers.

$$
\begin{array}{rcl}
\emptyset \Rightarrow \varphi & \equiv & \mathsf{tt} \\
D \Rightarrow c & \equiv & \mathsf{tt} \;\; ; \text{ if } D \subseteq c \\
D \Rightarrow ([a]\varphi) & \equiv & [a](D \Rightarrow \varphi) \\
D \Rightarrow (\varphi_1 \wedge \varphi_2) & \equiv & (D \Rightarrow \varphi_1) \wedge (D \Rightarrow \varphi_2) \\
D \Rightarrow (x \,\mathsf{in}\, \varphi) & \equiv & x \,\mathsf{in}\, (\{x\} D \Rightarrow \varphi) \\
D \Rightarrow (p \vee \varphi) & \equiv & p \vee (D \Rightarrow \varphi) \\
D \Rightarrow (c \vee \varphi) & \equiv & (D \wedge \neg c) \Rightarrow \varphi \\
D \Rightarrow (\mathbb{V} \varphi) & \equiv & \mathbb{V}(D^{\uparrow} \Rightarrow \varphi) \;\; ; \text{ if } D^{\downarrow} \subseteq D \\
D \Rightarrow X & \equiv & D \Rightarrow \mathcal{D}(X)
\end{array}
$$

Table 5: Constraint Propagation

Trivial Equation Elimination: Equations of the form $X \stackrel{def}{=} [a]X$ are easily seen to have $X = \mathsf{tt}$ as solution and may thus be removed. More generally, let S be the largest set of identifiers such that whenever $X \in S$ and $X \stackrel{def}{=} \varphi$ then $\varphi[\mathsf{tt}/S]$[7] can be simplified to tt. Then all identifiers of S can be removed provided the value tt is propagated to all uses of identifiers from S (as under Constant Propagation). The maximal set S may be efficiently computed using standard fixed point computation algorithms.

Equivalence Reduction: If two identifiers X and Y are semantically equivalent (i.e. are satisfied by the same timed transition systems) we may collapse them into a single identifier and thus obtain reduction. However, semantical equivalence is computationally very hard[8]. To obtain a cost effective strategy we approximate semantical equivalence of identifiers as follows: Let $\mathcal{R}$ be an equivalence relation on identifiers. $\mathcal{R}$ may be extended homomorphically to formulas in the obvious manner: i.e. $(\varphi_1 \wedge \varphi_2)\mathcal{R}(\vartheta_1 \wedge \vartheta_2)$ if $\varphi_1 \mathcal{R} \vartheta_1$ and $\varphi_2 \mathcal{R} \vartheta_2$, $(x \,\mathsf{in}\, \varphi)\mathcal{R}(x \,\mathsf{in}\, \vartheta)$ and $[a]\varphi \mathcal{R} [a]\vartheta$ if $\varphi \mathcal{R} \vartheta$ and so on. Now let $\cong$ be the maximal equivalence relation on identifiers such that whenever $X \cong Y$, $X \stackrel{def}{=} \varphi$ and $Y \stackrel{def}{=} \vartheta$ then $\varphi \cong \vartheta$. Then $\cong$ provides the desired cost effective approximation: whenever $X \cong Y$ then X and Y are indeed semantically equivalent. Moreover, $\cong$ may be efficiently computed using standard fixed point computation algorithms.

In the following examples we apply the above transformation strategies to the quotient formula obtained in Example 5.1. In particular, the strategies will find the quotient formula to be trivially true in certain cases.

Example 5.2 *Reconsider Example 5.1 with* Y_0, X_0 *and* X_1 *abbreviating* $Y/_f\, k_0$, $X/_f\, k_0$ *and* $X/_f\, k_1$. *Now* Y_0 *is the sufficient and necessary requirement to* A_n *in order that*

[7] $\varphi[\mathsf{tt}/S]$ is the formula obtained by substituting all occurrences of identifiers from S in φ with the formula tt.

[8] For the full logic T_μ the equivalence problem is undecidable.

$$
\begin{array}{rcl}
(D_0 \Rightarrow X_0) & \equiv & [b]\Big(y \text{ in } (D_0 \Rightarrow X_1)\Big) \wedge \Forall\Big({D_0}^{\uparrow} \Rightarrow X_0\Big) \\
({D_0}^{\uparrow} \Rightarrow X_0) & \equiv & [b]\Big(y \text{ in } (D_1 \Rightarrow X_1)\Big) \wedge \Forall\Big({D_0}^{\uparrow} \Rightarrow X_0\Big) \\
(D_1 \Rightarrow X_1) & \equiv & \Big((D_1 \wedge y \geq n) \Rightarrow [c]\mathsf{ff}\Big) \wedge \Forall({D_1}^{\uparrow} \Rightarrow X_1) \\
(D_0 \Rightarrow X_1) & \equiv & \Big((D_0 \wedge y \geq n) \Rightarrow [c]\mathsf{ff}\Big) \wedge \Forall({D_0}^{\uparrow} \Rightarrow X_1) \\
({D_0}^{\uparrow} \Rightarrow X_1) & \equiv & \Big(({D_0}^{\uparrow} \wedge z < i \wedge y \geq n) \Rightarrow [c]\mathsf{ff}\Big) \wedge \Forall(({D_0}^{\uparrow} \wedge z < i)^{\uparrow} \Rightarrow X_1) \\
({D_1}^{\uparrow} \Rightarrow X_1) & \equiv & \Big(({D_1}^{\uparrow} \wedge z < i \wedge y \geq n) \Rightarrow [c]\mathsf{ff}\Big) \wedge \Forall(({D_1}^{\uparrow} \wedge z < i)^{\uparrow} \Rightarrow X_1)
\end{array}
$$

Table 6: Equations after Constraint Propagation.

$A_n \mid_f B_n$ *satisfies* Y. *From the definition of satisfiability for timed automata we see that:*

$$A_n \models Y_0 \quad \textit{if and only if} \quad A_n \models \mathsf{tt} \Rightarrow \Big(y \text{ in } Y_0\Big)$$

This provides an initial basis for constraint propagation. Using the propagation laws from Table 5 we get:

$$\mathsf{tt} \Rightarrow \Big(y \text{ in } Y_0\Big) \;\equiv\; \mathsf{tt} \Rightarrow \Big(\{y,z\} \text{ in } X_0\Big) \;\equiv\; \{y,z\} \text{ in } \Big(D_0 \Rightarrow X_0\Big)$$

where $D_0 = (y = 0 \wedge z = 0)$. *This makes the implication* $D_0 \Rightarrow X_0$ *applicable to constraint propagation as follows:*

$$
\begin{array}{rcll}
(D_0 \Rightarrow X_0) & \equiv & D_0 \Rightarrow \Big[(z \geq i) \vee \Big([b](y \text{ in } X_1) \wedge \Forall X_0\Big)\Big] & \\
& \equiv & \Big(D_0 \Rightarrow [b](y \text{ in } X_1)\Big) \wedge \Big(D_0 \Rightarrow \Forall X_0\Big) & \textit{as } (z < i \wedge D_0) = D_0 \\
& \equiv & [b]\Big(y \text{ in } (D_0 \Rightarrow X_1)\Big) \wedge \Forall\Big({D_0}^{\uparrow} \Rightarrow X_0\Big) &
\end{array}
$$

Continuing constraint propagation yields the equations in Table 6, where $D_1 = (y = 0 \wedge z < i)$.

Example 5.3 (Example 5.2 Continued) *Now consider the case when* $n \geq i$. *That is the delay* n *of the component* B_n *exceeds the delay* i *required as a minimum by the property* Y. *Thus the component* B_n *ensures on its own the satisfiability of* Y; *i.e. for* any *choice of* A *the system* $A \mid_f B_n$ *will satisfy* Y. *In this particular case (i.e.* $n \geq i$*) it is easy to see that* $({D_i}^{\uparrow} \wedge z < i \wedge y \geq n) = \mathsf{ff}$ *for* $i = 0, 1$ *as* ${D_i}^{\uparrow}$ *ensures* $z \geq y$. *Also for* $i = 0, 1$, $(D_i \wedge y \geq n) = \mathsf{ff}$ *as* $D_i \Rightarrow y = 0$ *and we assume* $n > 0$. *Finally, it is easily seen that* $({D_i}^{\uparrow} \wedge z < i)^{\uparrow} = {D_i}^{\uparrow}$ *for* $i = 0, 1$. *Inserting these observations — which all may be efficiently computed — in the equations of Table 6 we get the simplified equations in Table 7.*

Now, the conjuncts $\mathsf{ff} \Rightarrow [c]\mathsf{ff}$ *are obviously equivalent to* tt *and will thus be removed by the boolean simplification transformations. Now, using our strategy for Trivial Equation Elimination, it may be found that all the equations in Table 7 are trivial and may consequently be removed (simplified to* tt*). To see this, simply observe that*

$$
\begin{array}{rcl}
(D_0 \Rightarrow X_0) & \equiv & [b]\Big(y \,\mathsf{in}\, (D_0 \Rightarrow X_1)\Big) \wedge \mathbb{V}\Big({D_0}^{\uparrow} \Rightarrow X_0\Big) \\
({D_0}^{\uparrow} \Rightarrow X_0) & \equiv & [b]\Big(y \,\mathsf{in}\, (D_1 \Rightarrow X_1)\Big) \wedge \mathbb{V}\Big({D_0}^{\uparrow} \Rightarrow X_0\Big) \\
(D_1 \Rightarrow X_1) & \equiv & \Big(\mathtt{ff} \Rightarrow [c]\mathtt{ff}\Big) \wedge \mathbb{V}({D_1}^{\uparrow} \Rightarrow X_1) \\
(D_0 \Rightarrow X_1) & \equiv & \Big(\mathtt{ff} \Rightarrow [c]\mathtt{ff}\Big) \wedge \mathbb{V}({D_0}^{\uparrow} \Rightarrow X_1) \\
({D_0}^{\uparrow} \Rightarrow X_1) & \equiv & \Big(\mathtt{ff} \Rightarrow [c]\mathtt{ff}\Big) \wedge \mathbb{V}({D_0}^{\uparrow} \Rightarrow X_1) \\
({D_1}^{\uparrow} \Rightarrow X_1) & \equiv & \Big(\mathtt{ff} \Rightarrow [c]\mathtt{ff}\Big) \wedge \mathbb{V}({D_1}^{\uparrow} \Rightarrow X_1)
\end{array}
$$

Table 7: Equations after Simplification.

substituting $\mathtt{tt}$ *for* $D_i \Rightarrow X_j$ *and* ${D_i}^{\uparrow} \Rightarrow X_j$ *on all right-hand sides in Table 7 leads to formulas which clearly can be simplified to* $\mathtt{tt}$*. Thus, in the case* $n \geq i$*, our minimisation heuristics will yield* $\mathtt{tt}$ *as the property required of* A *in order that* $A \,|_f\, B_n$ *satisfies* Y*.*

6 Conclusion and Future Work

In developing algorithms for automated verification of real-time systems modelled as networks of timed automata, we need to deal with two types of potential state-space explosions: explosion in the space of control nodes, and explosion in the region space over clock-variables. To attack these explosion problems, we have developed and combined compositional and symbolic model-checking techniques. The symbolic technique has been implemented in the verification tool UPPAAL.

We should point out that the safety logic presented in this paper is designed for efficient implementation. Though the logic is less expressive than the full version of the timed μ-calculus T_μ, it is expressive enough to specify safety properties as well as bounded liveness properties. As future work, we shall study the practical applicability of this logic and UPPAAL by further examples. Our experience shows that the practical limits of UPPAAL is caused by the space-complexity rather than the time-complexity of the model-checking algorithms. Thus, future work includes development of more space-efficient methods for representation and manipulation of clock constraints. For a verification tool to be of practical use in a design process it is of out most importance that the tool offers diagnostic information in case of erroneous. Based on the synthesis technique presented in [GL95] we intend to extend UPPAAL with the ability to generate diagnostic information.

Acknowledgements

The UPPAAL tool has been implemented in large parts by Johan Bengtsson and Fredrik Larsson. The authors would like to thank them for their excellent work. The first author would also like to thank Francois Laroussinie for several interesting

discussions on the subject of compositional model-checking. The last two authors want to thank the Steering Committee members of NUTEK, Bengt Asker and Ulf Olsson, for useful feedback on practical issues.

References

[ACD90] Rajeev Alur, Costas Courcoubetis, and David Dill. Model-checking for Real-Time Systems. In *Proc. of Logic in Computer Science*, pages 414–425. IEEE Computer Society Press, June 1990.

[AD94] R. Alur and D. Dill. Automata for Modelling Real-Time Systems. *Theoretical Computer Science*, 126(2):183–236, April 1994.

[And95] Henrik Reif Andersen. Partial Model Checking. In *Proc. of Symp. on Logic in Computer Science*, 1995.

[BCM+90] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic Model Checking: 10^{20} states and beyond. In *Proc. of IEEE Symp. on Logic in Computer Science*, 1990.

[Bel57] Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[BL96] Johan Bengtsson and Fredrik Larsson. UPPAAL a Tool for Automatic Verification of Real-time Systems. Master's thesis, Uppsala University, 1996. Available as `http://www.docs.uu.se/docs/rtmv/bl-report.pdf`.

[CFJ93] E. M. Clarke, T. Filkorn, and S. Jha. Exploiting Symmetry in Temporal Logic Model Checking. In *Proc. of the 5*th *Int. Conf. on Computer Aided Verification*, number 697 in Lecture Notes in Computer Science, 1993.

[CGL92] E. M. Clarke, O. Grümberg, and D. E. Long. Model Checking and Abstraction. *Principles of Programming Languages*, 1992.

[Dil89] David Dill. Timing Assumptions and Verification of Finite-State Concurrent Systems. In J. Sifakis, editor, *Proc. of Automatic Verification Methods for Finite State Systems*, number 407 in Lecture Notes in Computer Science, pages 197–212. Springer–Verlag, 1989.

[EJ93] E. A. Emerson and C. S. Jutla. Symmetry and Model Checking. In *Proc. of the 5*th *Int. Conf. on Computer Aided Verification*, number 697 in Lecture Notes in Computer Science, 1993.

[GL95] J.C. Godskesen and K.G. Larsen. Synthesizing Distinghuishing Formulae for Real Time Systems — Extended Abstract. In *Proc. of MFCS'95*, Lecture Notes in Computer Science, 1995.

[GW91] P. Godefroid and P. Wolper. A Partial Approach to Model Checking. In *Proc. of IEEE Symp. on Logic in Computer Science*, pages 406–415, 1991.

[Hal93] Nicolas Halbwachs. Delay Analysis in Synchronous Programs. In *Proc. of the 5*th *Int. Conf. on Computer Aided Verification*, number 697 in Lecture Notes in Computer Science, 1993.

[HL89] Hans Hüttel and Kim G. Larsen. The use of static constructs in a modal process logic. In *Logic at Botik'89*, number 363 in Lecture Notes in Computer Science, pages 163–180. Springer–Verlag, 1989.

[HNSY94] Thomas. A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic Model Checking for Real-Time Systems. *Information and Computation*, 111(2):193–244, 1994.

[LL95] Francois Laroussinie and Kim G. Larsen. Compositional Model Checking of Real Time Systems. In *Proc. of CONCUR '95: Concurrency Theory*, Lecture Notes in Computer Science. Springer–Verlag, 1995.

[LLW95] Francois Laroussinie, Kim G. Larsen, and Carsten Weise. From Timed Automata to Logic — and Back. In *Proc. of MFCS'95*, Lecture Notes in Computer Sciencie, 1995. Also BRICS report series RS–95–2.

[Tar55] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[Val90] A. Valmari. A Stubborn Attack on State Explosion. *Theoretical Computer Science*, 3, 1990.

[YPD94] Wang Yi, Paul Pettersson, and Mats Daniels. Automatic Verification of Real-Time Communicating Systems By Constraint-Solving. In Dieter Hogrefe and Stefan Leue, editors, *Proc. of the 7th Int. Conf. on Formal Description Techniques*, pages 223–238. North–Holland, 1994.

Paper C

Kim G. Larsen, Paul Pettersson, and Wang Yi. Diagnostic Model-Checking for Real-Time Systems. In Rajeev Alur, Thomas A. Henzinger and Eduardo D. Sontag, editors, *Proceedings of Workshop on Verification and Control of Hybrid Systems III*, number 1066 in Lecture Notes in Computer Science, pages 575–586. Springer–Verlag 1995.

Diagnostic Model-Checking for Real-Time Systems

Kim G. Larsen*, Paul Pettersson†, and Wang Yi†

* BRICS, Aalborg University, Denmark. Email: kgl@cs.auc.dk.
† Department of Computer Systems, Uppsala University, Sweden.
Email: {paupet,yi}@docs.uu.se.

Abstract. In this paper we present a diagnostic model-checking technique to facilitate debugging of system descriptions. The basic idea is to let the model-checker generate a sequence of transitions, namely a diagnostic trace, to explain or prove why certain states are reachable in a system description. The diagnostic traces turn out to be very useful for modelling. For example, they can be used in the subsequent debugging of an erroneous system description to locate error sources. In general, an iterative process can be adopted in which system descriptions are successively refined based on the output of earlier verifications. To illustrate the usefulness of the diagnostic feature, we present a case-study where UPPAAL is used to debug early versions of an audio-control protocol by Philips.

1 Introduction

In the area of model-checking for real-time systems, research effort has been focused on improving the efficiency of various model-checking procedures. The difficulty of *modelling*, i.e. constructing system descriptions, has been paid little attention by researchers in the community. Many users of the existing model-checkers, e.g. UPPAAL for timed system, may have experienced that a large part of the total time in most of the case studies with these tools is spent on modeling. The problem is that the system descriptions developed during the earlier phases of the case studies are often erroneous and it is difficult to find error sources without tool support.

In this paper, we address this problem by developing a diagnostic model-checking algorithm. It is based on the observation that whenever a state is reachable in a system description, there must be a sequence of transitions leading to the state, namely a *diagnostic trace*. Thus, the diagnostic trace is a proof that the reachability property is satisfied by the system description. We generalise this idea to more general logical formulas specified in a timed logic for specifying safety and bounded-liveness properties. More precisely, we develop a model-checking procedure that provides not only a negative answer, but also generates a diagnostic trace whenever a system description does not satisfy a given formula. Such a trace describes a scenario that can be performed by the system model, but which is undesired in the sense that it violates the currently checked property. The diagnostic trace can be used in the subsequent debugging of the system description to locate the error sources. Thus, an iterative process can be adopted in which system descriptions are successively refined based on the output of earlier verifications. If the diagnostic model-checking algorithm fails to produce a diagnostic trace, it can be concluded that the currently verified property is satisfied by the system description.

The diagnostic model-checking procedure has been implemented and added to the verification tool UPPAAL. To demonstrate its usefulness, we present a case-study where UPPAAL is applied to verify a version of Philips audio-control protocol [BPV94, HWT95, DY95]; a bus protocol by Philips, which is used in their audio and TV equipments to transmit control information between components. We first present an erroneous description of the protocol in the model of timed automata, that was derived from an early version of the description in [HWT95]. To debug the description, the diagnostic model-checking feature of UPPAAL is used to generate diagnostic traces which are analysed to find and remove errors. The process is iterated three times before the protocol description is found to satisfy the main correctness property of the protocol, i.e. the bit sequences received on the receiving node are always guaranteed to match the sent bit sequences.

The rest of this paper is organised as follows: In the next section we give a short review of the notions of timed automata and networks; in Section 3 a logic for safety and bounded liveness properties is presented. Section 4 describes the diagnostic model-checking procedure; in Section 5 we show how these results have been applied in a case study where Philips audio-control protocol was analysed.

2 Real-Time Systems

We shall use *timed transition systems* as a basic semantical model for real-time systems. The type of systems we are studying will be a particular class of timed transition systems that are syntactically described by *networks of timed automata* [YPD94, LPY95].

2.1 Timed Transition Systems

A timed transition system is a labeled transition system with two types of labels: atomic actions and delay actions (i.e. positive reals), representing discrete and continuous changes of real-time systems, respectively.

Let $\mathcal{A}$ be a finite set of actions and $\mathcal{P}$ be a set of atomic propositions. We use $\mathbb{R}_+$ to stand for the set of non-negative real numbers, Δ for the set of delay actions $\{\epsilon(d) \mid d \in \mathbb{R}_+\}$, and Σ for the union $\mathcal{A} \cup \Delta$.

Definition 2.1 (Timed Transition System) *A timed transition system over $\mathcal{A}$ and $\mathcal{P}$ is a tuple $\mathcal{S} = \langle S, s_0, \Leftrightarrow\!\!\rightarrow, V\rangle$, where S is a set of states, s_0 is the initial state, $\Leftrightarrow\!\!\rightarrow \subseteq S \times \Sigma \times S$ is a transition relation, and $V : S \to 2^{\mathcal{P}}$ is a proposition assignment function.* □

We will need for the transition relation $\Leftrightarrow\!\!\rightarrow$ to satisfy the following well-known properties (see e.g. [Yi91]):

- *(Time Determinism)* $s \overset{\epsilon(d)}{\Leftrightarrow\!\!\rightarrow} s_1$ and $s \overset{\epsilon(d)}{\Leftrightarrow\!\!\rightarrow} s_2 \Rightarrow s_1 = s_2$.
- *(Time Continuity)* $s \overset{\epsilon(d+e)}{\Leftrightarrow\!\!\rightarrow} s' \Leftrightarrow \exists s''.s \overset{\epsilon(d)}{\Leftrightarrow\!\!\rightarrow} s'' \overset{\epsilon(e)}{\Leftrightarrow\!\!\rightarrow} s'$.

Whenever defined, we will use the notation s^d for the state satisfying $s \overset{\epsilon(d)}{\Leftrightarrow} s^d$. Note that the state s^d is unique due to time determinism.

In order to study compositionality problems we introduce a parallel composition between timed transition systems. Following [HL89] we use synchronisation functions that generalise a large range of existing notions of parallel compositions. A *synchronisation function* f is a partial function $(\mathcal{A} \cup \{0\}) \times (\mathcal{A} \cup \{0\}) \hookrightarrow \mathcal{A}$, where 0 denotes a distinguished no-action symbol. Now, let $\mathcal{S}_i = \langle S_i, s_{i,0}, \Leftrightarrow_i, V_i \rangle$, $i = 1, 2$, be two timed transition systems and let f be a synchronisation function. Then the *parallel composition* $\mathcal{S}_1 \mid_f \mathcal{S}_2$ is the timed transition system $\langle S, s_0, \Leftrightarrow, V \rangle$, where $s_1 \mid_f s_2 \in S$ whenever $s_1 \in S_1$ and $s_2 \in S_2$, $s_0 = s_{1,0} \mid_f s_{2,0}$, $\Leftrightarrow$ is inductively defined as follows:

- $s_1 \mid_f s_2 \overset{c}{\Leftrightarrow} s_1' \mid_f s_2'$ if $s_1 \overset{a}{\Leftrightarrow}_1 s_1'$, $s_2 \overset{b}{\Leftrightarrow}_2 s_2'$ and $f(a, b) = c$
- $s_1 \mid_f s_2 \overset{\epsilon(d)}{\Leftrightarrow} s_1' \mid_f s_2'$ if $s_1 \overset{\epsilon(d)}{\Leftrightarrow}_1 s_1'$ and $s_2 \overset{\epsilon(d)}{\Leftrightarrow}_2 s_2'$

and finally, the proposition assignment function V is defined by $V(s_1 \mid_f s_2) = V_1(s_1) \cup V_2(s_2)$.

We now introduce the notion of a *trace*. A trace σ of a timed transition system is a finite alternating sequence of the form:

$$\sigma = s_0 \overset{\epsilon(d_0)}{\Leftrightarrow} s_0' \overset{a_1}{\Leftrightarrow} s_1 \overset{\epsilon(d_1)}{\Leftrightarrow} s_1' \overset{a_2}{\Leftrightarrow} s_2 \overset{\epsilon(d_2)}{\Leftrightarrow} \ldots \overset{a_n}{\Leftrightarrow} s_n \overset{\epsilon(d_n)}{\Leftrightarrow} s_n'$$

where $d_i \in \mathbb{R}_+$, i.e. a positive real number. A position π of a trace σ is a pair $\pi = (i, d)$ where $i \in 0 \ldots n$ and $0 \leq d \leq d_i$. We use $\mathcal{D}(\sigma, \pi)$ to stand for the accumulated delay of the trace σ before the position π, i.e. $\mathcal{D}(\sigma, \pi) = \sum_{j<i} d_j + d$ and $\sigma(\pi)$ for the suffix of σ starting from π, i.e.

$$\sigma(\pi) = s_i^d \overset{\epsilon(d_i - d)}{\Leftrightarrow} s_i' \overset{a_{i+1}}{\Leftrightarrow} s_{i+1} \overset{\epsilon(d_{i+1})}{\Leftrightarrow} \ldots \overset{a_n}{\Leftrightarrow} s_n \overset{\epsilon(d_n)}{\Leftrightarrow} s_n'$$

Whenever $s \overset{\alpha}{\Leftrightarrow} s_0$ ($\alpha \in \Sigma$) we shall denote by $s \overset{\alpha}{\Leftrightarrow} \sigma$ the trace obtained by extending σ[1]. We order positions lexicographically, denoted $\pi < \pi'$. Finally, we write $V(\sigma)$ for the set $V(s_0)$.

2.2 Networks of Timed Automata

A timed automaton [AD90] is a standard finite-state automaton extended with a finite collection of real-valued clocks. The clocks are assumed to proceed at the same rate and their values may be tested (compared with natural numbers) and reset (assigned to 0).

Definition 2.2 (Clock Constraints) *Let $\mathcal{C}$ be a set of real-valued clocks. We use $\mathcal{B}(\mathcal{C})$ to stand for the set of clock constraints ranged over by g, generated by the following syntax: $g ::= c \mid g \wedge g$, where c is an atomic constraint of the form: $x \sim n$ or $x \Leftrightarrow y \sim n$ for $x, y \in \mathcal{C}$, $\sim \in \{\leq, \geq, =, <, >\}$ and n is a natural number.* □

[1] In order to keep the extended trace alternating we might have to apply the time continuity property to avoid two neighboring delay-transitions and we might have to insert 0-delay transition in order to avoid neighboring action-transitions.

We shall use $\mathtt{tt}$ to stand for a constraint like $x \geq 0$ which is always true, and $\mathtt{ff}$ for a constraint $x < 0$ which is always false as clocks can only have non-negative values.

Definition 2.3 (Timed Automata) *A timed automaton A over actions $\mathcal{A}$, atomic propositions $\mathcal{P}$ and clocks $\mathcal{C}$ is a tuple $\langle N, l_0, \Leftrightarrow, I, V\rangle$, where N is a finite set of locations (control-locations), l_0 is the initial location, and $\Leftrightarrow \subseteq N \times \mathcal{B}(\mathcal{C}) \times \mathcal{A} \times 2^{\mathcal{C}} \times N$ corresponds to the set of edges. In the case, $\langle l, g, a, r, l'\rangle \in \Leftrightarrow$ we shall write, $l \overset{g\,a\,r}{\Leftrightarrow} l'$. $I : N \to \mathcal{B}(\mathcal{C})$ is a function which for each location assigns an invariant condition, and $V : N \to 2^{\mathcal{P}}$ is a proposition assignment function which for each location gives a set of atomic propositions true in the location.* □

A *state* of an automaton A is a pair (l, u) where l is a location of A and u a *clock assignment* for $\mathcal{C}$, mapping each clock in $\mathcal{C}$ to a value in $\mathbb{R}_+$. The initial state of A is (l_0, u_0) where u_0 is the initial clock assignment mapping all clocks in $\mathcal{C}$ to 0.

The semantics of A is given by the timed transition system $\mathcal{S}_A = \langle S, s_0, \Leftrightarrow, V\rangle$, where S is the set of states of A, s_0 is the initial state (l_0, u_0), $\Leftrightarrow$ is the transition relation defined as follows:

- $(l, u) \overset{a}{\Leftrightarrow} (l', u')$ if there exist r, g such that $l \overset{g\,a\,r}{\Leftrightarrow} l'$, g is satisfied by u and $u' = r[u]$ [2],
- $(l, u) \overset{\epsilon(d)}{\Leftrightarrow} (l', u')$ if $(l = l')$, $u' = u \oplus d$ [3] and $I(l')$ is satisfied by u',

and V is extended to S simply by $V(l, u) = V(l)$. We denote by $Tr(A)$ all traces of $\mathcal{S}_A$ starting from the initial state (l_0, u_0).

Parallel composition may now be extended to timed automata in the obvious way: for two timed automata A and B and a synchronisation function f, the parallel composition $A \mid_f B$ denotes the timed transition system $\mathcal{S}_A \mid_f \mathcal{S}_B$.

3 A Logic for Safety and Bounded Liveness Properties

We consider a timed modal logic to specify safety and bounded liveness properties. The logic may be seen as a fragment of the timed μ-calculus presented in [HNSY94], and also studied in [LLW95][4].

Definition 3.1 (Syntax) *Assume K is a finite set of clocks. Then formulas over K are defined by the following abstract syntax:*

$$\varphi \ ::= \ a \mid \varphi_1 \wedge \varphi_2 \mid a \vee \varphi \mid \mathsf{INV}(\varphi) \mid \varphi \ \mathsf{UNTIL}_r \ a$$

where $r \subseteq K$ and $a \ ::= \ c \mid p$ where c is an atomic clock constraint over K and $p \in \mathcal{P}$. □

[2] $r[u]$ is the assignment s.t. $r[u](x) = 0$ if $x \in r$ and $r[u](x) = u(x)$ otherwise.

[3] $(u \oplus d)$ is the assignment s.t. $(u \oplus d)(x) = u(x) + d$.

[4] The connectives of our logic are expressible as derived operators w.r.t. those in [LLW95].

$$
\begin{array}{rcl}
\sigma \models_v c & \textit{iff} & c(v) \\
\sigma \models_v p & \textit{iff} & p \in V(\sigma) \\
\sigma \models_v \varphi_1 \wedge \varphi_2 & \textit{iff} & \sigma \models_v \varphi_1 \text{ and } \sigma \models_v \varphi_2 \\
\sigma \models_v a \vee \varphi & \textit{iff} & \sigma \models_v a \text{ or } \sigma \models_v \varphi \\
\sigma \models_v \mathsf{INV}(\varphi) & \textit{iff} & \forall \pi : \sigma(\pi) \models_{v \oplus \mathcal{D}(\sigma,\pi)} \varphi \\
\sigma \models_v \varphi \ \mathsf{UNTIL}_r \ a & \textit{iff} & \left\{ \begin{array}{l} \forall \pi : \sigma(\pi) \models_{r[v] \oplus \mathcal{D}(\sigma,\pi)} \varphi, \\ \text{or} \\ \exists \pi : \Big(\sigma(\pi) \models_{r[v] \oplus \mathcal{D}(\sigma,\pi)} a \wedge \\ \qquad \forall \pi' < \pi : \sigma(\pi') \models_{r[v] \oplus \mathcal{D}(\sigma,\pi')} \varphi \Big) \end{array} \right.
\end{array}
$$

Table 1: Definition of Satisfiability.

Intuitively, for $\mathsf{INV}(\varphi)$ to be satisfied all reachable states must satisfy φ. $\varphi \ \mathsf{UNTIL}_r \ a$ is a weak until-property expressing that φ must either hold invariantly or until a. The use of the clock set r allows for *bounded liveness properties* to be expressed, e.g. $(x < 5) \ \mathsf{UNTIL}_{\{x\}} \ a$ insists that a must hold within 5 time units.

We interpret a formula φ with respect to a trace σ relative to a time assignment v over formula clocks K. We use $\sigma \models_v \varphi$ to mean that σ satisfies φ under v. The interpretation is defined on the structure of φ in Table 1. Naturally, if all the traces of an automaton satisfy a formula, we say that the automaton satisfies the formula.

Definition 3.2 (Diagnostic Trace) *Let $Tr(\varphi) = \{\sigma \mid \sigma \models_{v_o} \varphi\}$ where v_0 is the initial time assignment. For a timed automaton A and a formula φ we write $A \models \varphi$ when $Tr(A) \subseteq Tr(\varphi)$. If there exists a trace σ s.t. $\sigma \in Tr(A) \setminus Tr(\varphi)$, we write $A \not\models \varphi$ and in this case, σ is called a diagnostic trace of A w.r.t. φ.* □

4 Diagnostic Model-Checking

Given a network of timed automata A and a formula φ in the logic specifying a property, the so-called model-checking problem is to check if the formula is satisfied by the system. We will take an opposite point of view and check for $A \not\models \varphi$ instead of $A \models \varphi$. From a proof of $A \not\models \varphi$ we will then be able to synthesise a diagnostic trace which may prove useful in subsequent debugging. However, if we fail to prove $A \not\models \varphi$ we can assert that $A \models \varphi$.

4.1 Clock Constraints

To develop the diagnostic model-checking algorithm, we need a few operations to manipulate clock constraints. Given a clock constraint D, we shall call the set of clock assignments satisfying D, the *solution set* of D.

Definition 4.1 *Let A and A' be the solution sets of clock constraints $D, D' \in \mathcal{B}(\mathcal{C} \cup K)$. We define*

$$\begin{aligned} A \wedge A' &= \{w \mid w \in A \text{ and } w \in A'\} \\ A^{\uparrow} &= \{w + d \mid w \in A \text{ and } d \in \mathbb{R}_+\} \\ \{x\}A &= \{x[w] \mid w \in A\} \\ A{\downarrow}\mathcal{C} &= \{w{\downarrow}\mathcal{C} \mid w \in A\} \end{aligned}$$

where $w{\downarrow}\mathcal{C}$ denotes the restriction of w to the clock set $\mathcal{C}$. □

Note that the operations $A \wedge A'$, $A^{\uparrow}$ and $\{x\}A$ are defined as usual, see e.g. [YPD94, LPY95]. The solution set $A{\downarrow}\mathcal{C}$ is the restriction of A to the clocks in $\mathcal{C}$. We extend the reset-operator $\{x\}A$ to sets of clocks. Let $r = \{x_1, ..., x_n\}$ be a set of clocks. We define $r(A)$ recursively by $\{\}(A) = A$ and $\{x_1...x_n\}(A) = \{x_1\}(\{x_2...x_n\}A)$. The following proposition establishes that the class of clock constraints $\mathcal{B}(\mathcal{C} \cup K)$ is closed under the four operations defined above.

Proposition 4.1 (Closure Property of Clock Constraints) *Let $D, D' \in \mathcal{B}(\mathcal{C} \cup K)$ with solution sets A and A', and $x \in \mathcal{C} \cup K$. Then there exist $D_1, D_2, D_3, D_4 \in \mathcal{B}(\mathcal{C} \cup K)$ with solution sets $A^{\uparrow}$, $\{x\}A$, $A \wedge A'$ and $A{\downarrow}\mathcal{C}$ respectively.*

Proof: See [Dil89]. □

In order to save notation, from now on, we shall simply use $D^{\uparrow}$, $\{x\}D$ $D \wedge D'$ and $D{\downarrow}\mathcal{C}$ to denote the clock constraints which are guaranteed to exist due to the above proposition. Furthermore, we shall use $D^{\uparrow l}$ to denote $(D \wedge I(l))^{\uparrow} \wedge I(l)$ where $I(l)$ is the invariant condition of location l.

We will also need a few *predicates* over clock constraints for the diagnostic model-checking procedure. We write $D \subseteq D'$ to mean that the solution set of D is included in the solution set of D', $D = \emptyset$ to mean that the solution set of D is empty and $u \in D$ to denote that the time assignment u belongs to the solution set of D[5].

4.2 Model-Checking with Diagnostic Synthesis

Note that the definition $A \not\models \varphi$ means that there exists a trace σ of A such that $\sigma \notin Tr(\varphi)$. Intuitively, σ is a possible execution of A that does not meet the requirement φ, and therefore it may be used as diagnostic information for subsequent debugging. In order to effectively construct diagnostic traces, we define a relation $\not\vdash$ of the following type:

$$\sigma \not\vdash (l, D) : \varphi$$

where σ is a trace of automaton A over the automata clocks $\mathcal{C}$, l is a location of A, D is a clock constraint in $\mathcal{B}(\mathcal{C} \cup K)$ and φ is a formula over K. Now $\not\vdash$ is the smallest relation satisfying the rules of Table 2.

To exemplify the intuitive explanation of the inference rules we use the third invariant rule. The assertion $(l, u) \overset{a}{\Leftrightarrow} (l', u') \Leftrightarrow \cdots \not\vdash (l, D) : \mathsf{INV}(\varphi)$ can be justified if any of the symbolic states, reachable using an edge $l \overset{g\,a\,r}{\Leftrightarrow} l'$ from the symbolic state (l, D), does *not* satisfy the invariant property $\mathsf{INV}(\varphi)$. The clock

[5] We will also write $u \in D$ to mean the operation of computing a time assignment u given a constraint system D.

c	$\dfrac{}{(l,u) \longrightarrow \cdots \not\forall\ (l,D) : c} \quad w \in D \wedge \neg c, u = w{\downarrow}\mathcal{C}$
p	$\dfrac{}{(l,u) \longrightarrow \cdots \not\forall\ (l,D) : p} \quad \begin{array}{l} w \in D,\ u = w{\downarrow}\mathcal{C}, \\ p \notin V(l) \end{array}$
$\varphi_1 \wedge \varphi_2$	$\dfrac{\sigma \not\forall\ (l,D) : \varphi_i}{\sigma \not\forall\ (l,D) : \varphi_1 \wedge \varphi_2} \quad i = 1 \text{ or } i = 2$
$a \vee \varphi$	$\dfrac{\sigma \not\forall\ (l, D \wedge \neg c) : \varphi}{\sigma \not\forall\ (l,D) : c \vee \varphi} \qquad \dfrac{\sigma \not\forall\ (l,D) : \varphi}{\sigma \not\forall\ (l,D) : p \vee \varphi}\ p \notin V(l)$
$\mathsf{INV}(\varphi)$	$\dfrac{\sigma \not\forall\ (l,D) : \varphi}{\sigma \not\forall\ (l,D) : \mathsf{INV}(\varphi)}$ $\dfrac{(l,u') \longrightarrow \cdots \not\forall\ (l, D^{\uparrow l}) : \mathsf{INV}(\varphi)}{(l,u) \xrightarrow{\epsilon(d)} (l,u') \cdots \not\forall\ (l,D) : \mathsf{INV}(\varphi)} \quad \begin{array}{l} u \in D{\downarrow}\mathcal{C}, \\ u' = u \oplus d \end{array}$ $\dfrac{(l',u') \longrightarrow \cdots \not\forall\ (l', r(g \wedge D)) : \mathsf{INV}(\varphi)}{(l,u) \xrightarrow{a} (l',u') \longrightarrow \cdots \not\forall\ (l,D) : \mathsf{INV}(\varphi)} \quad \begin{array}{l} l \xrightarrow{g\ a\ r} l',\ u' = r[u], \\ u \in (g \wedge D){\downarrow}\mathcal{C} \end{array}$
$\varphi \mathsf{UNTIL}_r a$	$\dfrac{\sigma \not\forall\ (l, r(D)) : \varphi \mathsf{UNTIL}_\emptyset\ a}{\sigma \not\forall\ (l,D) : \varphi \mathsf{UNTIL}_r\ a}$
$\varphi \mathsf{UNTIL}_\emptyset c$	$\dfrac{\sigma \not\forall\ (l, D \wedge \neg c) : \varphi}{\sigma \not\forall\ (l,D) : \varphi\ \mathsf{UNTIL}_\emptyset\ c}$ $\dfrac{(l,u') \longrightarrow \cdots \not\forall\ (l, (D \wedge \neg c)^{\uparrow l}) : (\varphi\ \mathsf{UNTIL}_\emptyset\ c)}{(l,u) \xrightarrow{\epsilon(d)} (l,u') \longrightarrow \cdots \not\forall\ (l,D) : (\varphi\ \mathsf{UNTIL}_\emptyset\ c)} \quad \begin{array}{l} u \in (D \wedge \neg c){\downarrow}\mathcal{C}, \\ u' = u \oplus d \end{array}$ $\dfrac{(l',u') \longrightarrow \cdots \not\forall\ (l, r(g \wedge D \wedge \neg c)) : (\varphi\ \mathsf{UNTIL}_\emptyset\ c)}{(l,u) \xrightarrow{a} (l',u') \longrightarrow \cdots \not\forall\ (l,D) : (\varphi\ \mathsf{UNTIL}_\emptyset\ c)} \quad \begin{array}{l} l \xrightarrow{g\ a\ r} l',\ u' = r[u], \\ u \in (D \wedge g \wedge \neg c){\downarrow}\mathcal{C} \end{array}$
$\varphi \mathsf{UNTIL}_\emptyset p$	$\dfrac{\sigma \not\forall\ (l,D) : \varphi}{\sigma \not\forall\ (l,D) : \varphi\ \mathsf{UNTIL}_\emptyset\ p} \quad p \notin V(l)$ $\dfrac{(l,u') \longrightarrow \cdots \not\forall\ (l, D^{\uparrow l}) : (\varphi\ \mathsf{UNTIL}_\emptyset\ p)}{(l,u) \xrightarrow{\epsilon(d)} (l,u') \longrightarrow \cdots \not\forall\ (l,D) : (\varphi\ \mathsf{UNTIL}_\emptyset\ p)} \quad \begin{array}{l} p \notin V(l),\ u \in D{\downarrow}\mathcal{C}, \\ u' = u \oplus d \end{array}$ $\dfrac{(l',u') \longrightarrow \cdots \not\forall\ (l, r(g \wedge D)) : (\varphi\ \mathsf{UNTIL}_\emptyset\ p)}{(l,u) \xrightarrow{a} (l',u') \longrightarrow \cdots \not\forall\ (l,D) : (\varphi\ \mathsf{UNTIL}_\emptyset\ p)} \quad \begin{array}{l} p \notin V(l),\ l \xrightarrow{g\ a\ r} l', \\ u' = r[u],\ u \in (D \wedge g){\downarrow}\mathcal{C} \end{array}$

Table 2: Inference Rules for $\not\forall$.

assignments in the resulting symbolic state is restricted to the (non-empty) constraint system $r(g \wedge D)$. The premise of the rule assumes the existence of a diagnostic trace for $(l', r(g \wedge D)) : \mathsf{INV}(\varphi)$ and the side-condition of the rule provides information as to how one may extend this trace (obviously with an a-transition) in order to obtain a diagnostic trace for $(l, D) : \mathsf{INV}(\varphi)$.

The rules in Table 2 are sound and complete in the following sense:

Theorem 4.1 *Assume A is a timed automaton with initial location l_0 and clock assignment u_0, and further v_0 is a clock assignment over K. Then:*

1. *whenever $\sigma \in Tr(A)$ and $\sigma \not\vdash (l_0, \{u_0 v_0\}) : \varphi$ then $\neg(\sigma \models_{v_0} \varphi)$*
2. *whenever $A \not\models \varphi$ then $\sigma \not\vdash (l_0, \{u_0 v_0\}) : \varphi$ for some $\sigma \in Tr(A)$*

where $\{u_0 v_0\}$ is the clock constraint with $u_0 v_0$ being the only solution.

Proof: The proof is by structural induction over φ. It is given in Appendix B. □

4.3 Towards an Algorithm

Given a symbolic state (l, D) of the automata A and a property φ it is decidable whether there exists a diagnostic trace σ such that $\sigma \not\vdash (l, D) : \varphi$. We obtain an algorithm by using the rules in Table 2 in two phases.

In the first phase, a goal directed search starting in the symbolic state (l, D) searching for a violating symbolic state, is performed, by using the inference rules in Table 2. We have the following two termination criteria for the symbolic state (l_n, D_n) and the property φ_n:

- *Success:* c or p axiom can be applied,
- *Fail:* for some i, $l_n = l_i$, $D_n \subseteq D_i$ and $\varphi_n = \varphi_i$.

The search will be terminated on the Fail criterion if all the possibilities of backtracking have been exhausted. It can then be asserted that the automaton A in any state complying with (l, D) satisfies φ. However, if the first phase terminates on the Success criterion it follows that $\sigma \not\vdash (l, D) : \varphi$. The rules in Table 2 provide a way to synthesise the diagnostic trace of the conclusion from a diagnostic trace of the premise, constituting the second phase. Note that, if the search in the first phase is performed using a breadth-first strategy, a resulting trace will be a *shortest* diagnostic trace.

4.4 An Example: Fischer's Mutual Exclusion Protocol

To illustrate the algorithm described in the previous sections we apply it to a version of Fischer's mutual exclusion protocol [AL92, YPD94]. Consider the timed automata shown in Figure 1. The two automata P1 and P2 model processes, and V a shared variable. Each process P1 and P2 has a critical section, represented by the locations CS1 and CS2 respectively.

Assume that the system is described by the following network of timed automata: ((P1|P2)||V), where | and || are the full interleaving and full synchronisation operators,

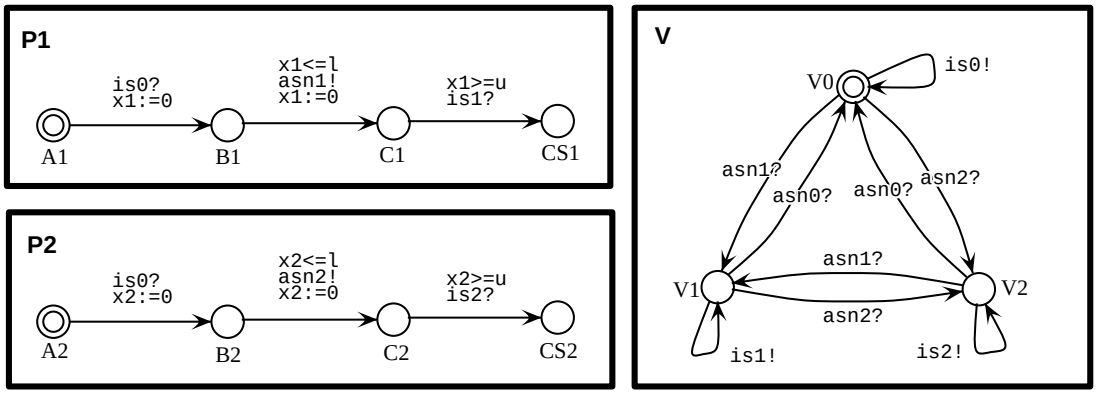


Figure 1: Fischer's Mutual Exclusion Protocol

induced by the synchronisation functions f and g respectively, defined by $f(0,a) = a, f(a,0) = a$, $g(a!,a?) = a$ and $g(a?,a!) = a$. Moreover, assume that $\neg\mathsf{at}(l)$ is an atomic proposition meaning that the system is not operating in control-location l, i.e. $\mathsf{at}(l') \in V(l)$ if $l = l'$ and $\neg\mathsf{at}(l') \in V(l)$ if $l' \neq l$ for all locations l and l'. We want to verify that there will never be more than one process in its critical section. This requirement can be formalised as the invariant property: $\mathsf{INV}(\mathit{Excl})$ where $\mathit{Excl} \stackrel{def}{=} \neg\mathsf{at}(\mathsf{CS1}) \vee \neg\mathsf{at}(\mathsf{CS2})$.

Under the assumption that $\mathtt{l} \geq \mathtt{u}$ we may use the inference rules in Table 2 to synthesise a trace σ of the system $((\mathsf{P1}|\mathsf{P2})||\mathsf{V})$ that is not permitted by the property $\mathsf{INV}(\mathit{Excl})$. Thus, under the assumption $\mathtt{l} \geq \mathtt{u}$, the mutual exclusion property does not hold.

$((\mathsf{CS1},\mathsf{CS2},\mathsf{V2}),(2*\mathtt{u},\mathtt{u}))$	$\not\models ((\mathsf{CS1},\mathsf{CS2},\mathsf{V2}),(-\mathtt{u} \leq \mathtt{x1}-\mathtt{x2} \leq \mathtt{l}, \mathtt{x2} \geq \mathtt{u})) : \mathit{Excl}$
$((\mathsf{CS1},\mathsf{C2},\mathsf{V2}),(2*\mathtt{u},\mathtt{u})) \stackrel{\mathtt{is2}}{\longrightarrow} \cdots$	$\not\models ((\mathsf{CS1},\mathsf{C2},\mathsf{V2}),(-\mathtt{u} \leq \mathtt{x1}-\mathtt{x2} \leq \mathtt{l})) : \mathsf{INV}(\mathit{Excl})$
$((\mathsf{CS1},\mathsf{C2},\mathsf{V2}),(\mathtt{u},0)) \stackrel{\epsilon(\mathtt{u})}{\longrightarrow} \cdots$	$\not\models ((\mathsf{CS1},\mathsf{C2},\mathsf{V2}),(\mathtt{u} \leq \mathtt{x1} \leq \mathtt{l}, \mathtt{x2} = 0)) : \mathsf{INV}(\mathit{Excl})$
$((\mathsf{CS1},\mathsf{B2},\mathsf{V1}),(\mathtt{u},\mathtt{u})) \stackrel{\mathtt{asn2}}{\longrightarrow} \cdots$	$\not\models ((\mathsf{CS1},\mathsf{B2},\mathsf{V1}),(\mathtt{x1} \geq \mathtt{u}, 0 \leq \mathtt{x2}-\mathtt{x1} \leq \mathtt{l})) : \mathsf{INV}(\mathit{Excl})$
$((\mathsf{CS1},\mathsf{B2},\mathsf{V1}),(\mathtt{u},\mathtt{u})) \stackrel{\epsilon(0)}{\longrightarrow} \cdots$	$\not\models ((\mathsf{CS1},\mathsf{B2},\mathsf{V1}),(\mathtt{x1} \geq \mathtt{u}, 0 \leq \mathtt{x2}-\mathtt{x1} \leq \mathtt{l})) : \mathsf{INV}(\mathit{Excl})$
$((\mathsf{C1},\mathsf{B2},\mathsf{V1}),(\mathtt{u},\mathtt{u})) \stackrel{\mathtt{is1}}{\longrightarrow} \cdots$	$\not\models ((\mathsf{C1},\mathsf{B2},\mathsf{V1}),(0 \leq \mathtt{x2}-\mathtt{x1} \leq \mathtt{l})) : \mathsf{INV}(\mathit{Excl})$
$((\mathsf{C1},\mathsf{B2},\mathsf{V1}),(0,0)) \stackrel{\epsilon(\mathtt{u})}{\longrightarrow} \cdots$	$\not\models ((\mathsf{C1},\mathsf{B2},\mathsf{V1}),(\mathtt{x1}=0, \mathtt{x2} \leq \mathtt{l})) : \mathsf{INV}(\mathit{Excl})$
$((\mathsf{B1},\mathsf{B2},\mathsf{V0}),(0,0)) \stackrel{\mathtt{asn1}}{\longrightarrow} \cdots$	$\not\models ((\mathsf{B1},\mathsf{B2},\mathsf{V0}),(\mathtt{x2} \leq \mathtt{x1})) : \mathsf{INV}(\mathit{Excl})$
$((\mathsf{B1},\mathsf{B2},\mathsf{V0}),(0,0)) \stackrel{\epsilon(0)}{\longrightarrow} \cdots$	$\not\models ((\mathsf{B1},\mathsf{B2},\mathsf{V0}),(\mathtt{x2}=0)) : \mathsf{INV}(\mathit{Excl})$
$((\mathsf{B1},\mathsf{A2},\mathsf{V0}),(0,0)) \stackrel{\mathtt{is0}}{\longrightarrow} \cdots$	$\not\models ((\mathsf{B1},\mathsf{A2},\mathsf{V0}),(\mathtt{x1} \leq \mathtt{x2})) : \mathsf{INV}(\mathit{Excl})$
$((\mathsf{B1},\mathsf{A2},\mathsf{V0}),(0,0)) \stackrel{\epsilon(0)}{\longrightarrow} \cdots$	$\not\models ((\mathsf{B1},\mathsf{A2},\mathsf{V0}),(\mathtt{x1}=0)) : \mathsf{INV}(\mathit{Excl})$
$((\mathsf{A1},\mathsf{A2},\mathsf{V0}),(0,0)) \stackrel{\mathtt{is0}}{\longrightarrow} \cdots$	$\not\models ((\mathsf{A1},\mathsf{A2},\mathsf{V0}),(\mathtt{x1} = \mathtt{x2})) : \mathsf{INV}(\mathit{Excl})$
$((\mathsf{A1},\mathsf{A2},\mathsf{V0}),(0,0)) \stackrel{\epsilon(0)}{\longrightarrow} \cdots$	$\not\models ((\mathsf{A1},\mathsf{A2},\mathsf{V0}),(\mathtt{x1} = \mathtt{x2} = 0)) : \mathsf{INV}(\mathit{Excl})$

From the above we obtain the following diagnostic trace:

$$\begin{aligned}\sigma = \quad & ((\mathsf{A1},\mathsf{A2},\mathsf{V0}),(0,0)) \stackrel{\mathtt{is0}}{\longrightarrow} ((\mathsf{B1},\mathsf{A2},\mathsf{V0}),(0,0)) \stackrel{\mathtt{is0}}{\longrightarrow} ((\mathsf{B1},\mathsf{B2},\mathsf{V0}),(0,0)) \stackrel{\mathtt{asn1}}{\longrightarrow} \\ & ((\mathsf{C1},\mathsf{B2},\mathsf{V1}),(0,0)) \stackrel{\epsilon(\mathtt{u})}{\longrightarrow} ((\mathsf{C1},\mathsf{B2},\mathsf{V1}),(\mathtt{u},\mathtt{u})) \stackrel{\mathtt{is1}}{\longrightarrow} ((\mathsf{CS1},\mathsf{B2},\mathsf{V1}),(\mathtt{u},\mathtt{u})) \stackrel{\mathtt{asn2}}{\longrightarrow} \\ & ((\mathsf{CS1},\mathsf{C2},\mathsf{V2}),(\mathtt{u},0)) \stackrel{\epsilon(\mathtt{u})}{\longrightarrow} ((\mathsf{CS1},\mathsf{C2},\mathsf{V2}),(2*\mathtt{u},\mathtt{u})) \stackrel{\mathtt{is2}}{\longrightarrow} ((\mathsf{CS1},\mathsf{CS2},\mathsf{V2}),(2*\mathtt{u},\mathtt{u}))\end{aligned}$$

which clearly is not allowed by the property $\mathsf{INV}(\mathit{Excl})$. Observe that for this particular problem, if $\mathtt{l} < \mathtt{u}$ Phase 1 ends in the symbolic state $((\mathsf{CS1}, \mathsf{C2}, \mathsf{V2}), (\mathtt{u} \leq \mathtt{x1} \leq \mathtt{l}, \mathtt{x2} = 0))$ (and consequently no proof for $\not\vdash$ can be obtained) since the clock constraint system is then empty. In fact, the protocol is correct when $\mathtt{l} < \mathtt{u}$.

5 Applications

The techniques presented in previous sections have been implemented in the verification tool UPPAAL. In this section, we demonstrate the usefulness of the diagnostic model-checking feature of UPPAAL by applying the tool in a case-study where early descriptions of Philips audio-control protocol [BPV94, HWT95] are debugged and verified.

5.1 UPPAAL

UPPAAL[6] is a symbolic model-checker for networks of timed automata. The current version[7], as well as the ongoing work of extending the tool kit, is implemented in C++. UPPAAL has a graphical interface (Autograph) allowing systems to be defined by drawing. The graphical definition may be automatically compiled into a textual format which also serves as a basic programming language for networks of timed automata. On the textual representation a number of useful syntactical checks may be done. The tool is based on the techniques described in previous sections but there is a number of extensions and limitations in the current implementation[7].

In UPPAAL, state invariants are derived from the enabling conditions of the outgoing edges of a state. Intuitively, the derived state invariant implements a maximal progress assumption where the control-state has to be switched, if possible, to another control-state before all enabling conditions become false. UPPAAL implements one action function, namely the action function f of complementary actions defined by $f(a!, a?) = a$ and $f(a?, a!) = a$.

UPPAAL has been extended beyond the techniques presented in previous sections. The tool is able to compile *non-zero constant slope hybrid systems* and *non-zero linear hybrid systems* to timed automata using the technique in [OSY94]. Another extension in UPPAAL is auxiliary data variables (i.e. variables not effected by delay-transitions). One such kind of auxiliary variables, that will be used later in this section, are integer-variables.

5.2 Philips Audio-Control Protocol

This protocol by Philips was first verified by Bosscher et.al. [BPV94] and recently using verification tools [HWT95]. The protocol is used for exchanging control information between components in modern audio equipment. Bit streams are encoded using Manchester encoding that relies on timing delay between signals. The protocol

[6] Information about UPPAAL is available on the web site `http://www.docs.uu.se/uppaal/`.

[7] The current UPPAAL version is 0.3, May 1995.

uses bit slots of four time units, a 1 bit is encoded by raising the voltage from low to high in the middle of the bit slot. A 0 bit is encoded in the opposite way. The goal of the protocol is to guarantee reliable communication if the timing error is bound to $\pm 5\%$ in all components. The communication is further complicated since voltage changes from high to low can not be reliably detected. The decoding has to be done using only the changing from low to high. A linear hybrid automaton network description of the protocol is shown in Figure 8.

To perform experiments on the protocol we used an early draft version of a description by Wong–Toi and Ho [HWT95][8]. In their work they automatically verify the audio-control protocol using the tool HyTech (The Cornell Hybrid Technology Tool is a symbolic model checker for linear hybrid systems). By reusing their description we avoid the difficult and time-consuming work of modelling the protocol.

The protocol is modeled as a parallel composition of four processes described below. Several integer variables are used for recording information: `leng` for recording the number of bits generated by the input automaton but not yet acknowledged as being received, `c` for representing the binary encoding of these bits, `k` for recording the parity of the number of bits generated, and `m` for recording the parity of the number of bits received. The four parallel processes are:

Input The Input automaton non-deterministically generates valid bit sequences for the Sender automaton. Valid bit sequences are restricted to either odd length or ending in two 0 bits. The Input also updates the shared integer variables `k`, `c` and `leng`, which are used by the Output_Ack automaton (see below).

Sender This automaton encodes the bit sequences by reading the value of the next bit from the Input automaton and determine the time delay for the next high voltage, modeled as an `up!`-action.

Receiver The Receiver automaton decodes the bit stream by measuring the time delay between two subsequent `up?`-actions received from the Sender. The decoded bits are then acknowledged by synchronising on the `output_1` or `output_0` port with the output-acknowledgment automaton. The Receiver also records the parity of the received number of bits by updating `m`.

Output_Ack The output-acknowledgment automaton checks the current number of unacknowledged bits (`leng`) together with their binary encoding (`c`) and acknowledges the bits decoded by the receiver. It also updates the values of the variables `leng` and `c`.

The way the protocol has been modeled enables the correctness properties to be verified by reachability analysis. By introducing the edge stop $\overset{\texttt{leng}\geq 1}{\Leftrightarrow}$ error in the receiver automaton, the received bit stream is guaranteed to be identical to the sent bit stream precisely when the system satisfies the property INV($\neg$at(*error*)).

[8] Available, at that time, from the Web server at Cornell University (`http://www.cs.cornell.edu/`).

$$
\begin{array}{l}
((\mathsf{start},\mathsf{start},\mathsf{start},\mathsf{ack}),(0,0),(0,0,0,0)) \xrightarrow{0} \\
((\mathsf{head\_is\_1},\mathsf{start},\mathsf{start},\mathsf{ack}),(0,0),(1,0,0,1)) \xrightarrow{\mathtt{input\_1}} \\
((\mathsf{head\_is\_0},\mathsf{rise\_1},\mathsf{start},\mathsf{ack}),(0,0),(2,1,0,2)) \xrightarrow{\mathtt{up}} \\
((\mathsf{head\_is\_0},\mathsf{transhigh},\mathsf{up\_1},\mathsf{ack}),(0,0),(2,1,1,2)) \xrightarrow{\mathtt{head\_0}} \\
((\mathsf{head\_is\_0},\mathsf{tranhigh\_0a},\mathsf{up\_1},\mathsf{ack}),(0,0),(2,1,1,2)) \xrightarrow{\mathtt{output\_1}} \\
((\mathsf{head\_is\_0},\mathsf{tranhigh\_0a},\mathsf{last\_is\_1},\mathsf{ack}),(0,0),(0,1,1,1)) \xrightarrow{\epsilon(76)} \\
((\mathsf{head\_is\_0},\mathsf{tranhigh\_0a},\mathsf{last\_is\_1},\mathsf{ack}),(76,76),(0,1,1,1)) \xrightarrow{\mathtt{input\_0}} \\
((\mathsf{head\_is\_1},\mathsf{translow},\mathsf{last\_is\_1},\mathsf{ack}),(0,76),(1,0,1,2)) \xrightarrow{\mathtt{head\_1}} \\
((\mathsf{head\_is\_1},\mathsf{translow\_1a},\mathsf{last\_is\_1},\mathsf{ack}),(0,76),1,0,1,2)) \xrightarrow{\epsilon(76)} \\
((\mathsf{head\_is\_1},\mathsf{translow\_1a},\mathsf{last\_is\_1},\mathsf{ack}),(76,152),(1,0,1,2)) \xrightarrow{0} \\
((\mathsf{head\_is\_1},\mathsf{rise\_1},\mathsf{last\_is\_1},\mathsf{ack}),(0,152),(1,0,1,2)) \xrightarrow{\mathtt{up}} \\
((\mathsf{head\_is\_1},\mathsf{transhigh},\mathsf{up\_0},\mathsf{ack}),(0,0),(1,0,0,2)) \xrightarrow{\mathtt{output\_neq\_0}} \\
((\mathsf{head\_is\_1},\mathsf{transhigh},\mathsf{error},\mathsf{ack}),(0,0),(1,0,0,2))
\end{array}
$$

Figure 2: Diagnostic trace from the first version of the protocol.

5.2.1 The First Version

The first version, shown in Figure 5, was an adjusted version of the description in [HWT95]. The adjustments were necessary due to differences in HyTech and Uppaal. This step comprised: transforming the invariant conditions of the original description into enabling conditions of the model in Uppaal, introducing complementary synchronisation actions, adding the edge stop $\overset{\mathtt{leng}\geq 1}{\Leftrightarrow}$ error in the receiver automaton, and model the modulo-2 counters `m` and `k` as integer variables. Modulo-2 addition $\oplus$ was modeled as a conditional value assignment on integers (e.g. `m==0, m:=1` or `m==1, m:=0`). This first version was also free from some obvious typing errors found in the original description of the system, e.g. the enabling condition `leng==leng+1` originally on the edge head_is_0 $\Leftrightarrow$ endeven_00 in the Input automaton will never be enabled and should be removed.

The protocol was then attempted verified but found erroneous[9]. Using the diagnostic traces, automatically synthesised by Uppaal, as debugging information the system was further improved. The trace is shown in Figure 2 [10][11]. The trace indicates errors in several ways. First recall that the existence of a trace imply that the correctness property is not satisfied. This particular trace is wrong since a `head_1`-action is followed by a subsequent `up`-action without an `input_1`-action in between. Also from the diagnostic trace in Figure 2, it was revealed that the action labels `output_neq_1?` and `output_neq_0?` was swapped in the Output_Ack automaton. This must be the case since `c` = 1 and `leng` = 2 implies that the next output should be 0

[9] Uppaal, installed on a SPARCstation 10, performs the attempted verification and reports a diagnostic trace in 2.2 seconds.

[10] The states are shown in this trace as triples, where the first component is the control-location, the second component is the clock assignment for the clocks `x` and `y`, and the third component is the value assignment for the auxiliary variables `c`, `k`, `m` and `leng`.

[11] This is a trace of the transformed version of the description, where the non-zero linear hybrid automata have been compiled into timed automata.

$$
\begin{array}{l}
((\mathsf{start},\mathsf{start},\mathsf{start},\mathsf{ack}),(0,0),(0,0,0,0)) \xrightarrow{0} \\
((\mathsf{head\_is\_1},\mathsf{start},\mathsf{start},\mathsf{ack}),(0,0),(1,0,0,1)) \xrightarrow{\texttt{input\_1}} \\
((\mathsf{head\_is\_0},\mathsf{rise\_1},\mathsf{start},\mathsf{ack}),(0,0),(2,1,0,2)) \xrightarrow{\texttt{up}} \\
((\mathsf{head\_is\_0},\mathsf{transhigh},\mathsf{up\_1},\mathsf{ack}),(0,0),(2,1,1,2)) \xrightarrow{\texttt{head\_0}} \\
((\mathsf{head\_is\_0},\mathsf{tranhigh\_0a},\mathsf{up\_1},\mathsf{ack}),(0,0),(2,1,1,2)) \xrightarrow{\texttt{output\_1}} \\
((\mathsf{head\_is\_0},\mathsf{tranhigh\_0a},\mathsf{last\_is\_1},\mathsf{ack}),(0,0),(0,1,1,1)) \xrightarrow{\epsilon(76)} \\
((\mathsf{head\_is\_0},\mathsf{tranhigh\_0a},\mathsf{last\_is\_1},\mathsf{ack}),(76,76),(0,1,1,1)) \xrightarrow{\texttt{input\_0}} \\
((\mathsf{endeven\_00},\mathsf{translow},\mathsf{last\_is\_1},\mathsf{ack}),(0,76),(0,1,1,2)) \xrightarrow{\texttt{head\_0}} \\
((\mathsf{endeven\_00},\mathsf{translow\_0},\mathsf{last\_is\_1},\mathsf{ack}),(0,76),(0,1,1,2)) \xrightarrow{\epsilon(38)} \\
((\mathsf{endeven\_00},\mathsf{translow\_0},\mathsf{last\_is\_1},\mathsf{ack}),(38,114),(0,1,1,2)) \xrightarrow{\texttt{head\_0}} \\
((\mathsf{endeven\_00},\mathsf{rise\_0},\mathsf{last\_is\_1},\mathsf{ack}),(0,114),(0,1,1,2)) \xrightarrow{\texttt{up}} \\
((\mathsf{endeven\_00},\mathsf{transhigh\_0},\mathsf{next\_is\_01},\mathsf{ack}),(0,0),(0,1,1,2)) \xrightarrow{\texttt{output\_0}} \\
((\mathsf{endeven\_00},\mathsf{transhigh\_0},\mathsf{up\_1},\mathsf{ack}),(0,0),(0,1,1,1)) \xrightarrow{\texttt{output\_neq\_1}} \\
((\mathsf{endeven\_00},\mathsf{transhigh\_0},\mathsf{error},\mathsf{ack}),(0,0),(0,1,1,1))
\end{array}
$$

Figure 3: Diagnostic trace from the first improved version of the protocol.

while `output_neq_0?` is signaled to acknowledge that the next output can *not* be 0.

5.2.2 The First Improved Version

The first improved version of the protocol is shown in Figure 6. In the figure, changes from the previous version have been marked by using bold faces. The missing actions `input_1?` on edges translow_1 $\Leftrightarrow\!\!\rightarrow$ rise_1 and translow_1a $\Leftrightarrow\!\!\rightarrow$ rise_1 in the Sender automaton have been added. Furthermore, the action labels `output_neq_0?` and `output_neq_1?` in the Output_Ack automaton have been swapped.

Once again, we attempted to verify the system; the systems was found erroneous. From the diagnostic trace shown in Figure 3, a timing error was discovered in the receiver automaton. In the control-state (endeven_00,transhigh_0,up_1,ack) the Receiver automaton has decoded a 1 bit but this is not the bit sent by the sender. The disagreement is monitored by the output acknowledgment automaton that makes the system violate the correctness property by offering an `output_neq_1?`-action. The reason for this error was found on the edges last_is_1 $\Leftrightarrow\!\!\rightarrow$ next_is_01 and last_is_1 $\Leftrightarrow\!\!\rightarrow$ up_0 where the enabling conditions on clock `y` was swapped.

5.2.3 The Second Improved Version

An even further improved version, shown in Figure 7, was prepared by swapping the enabling conditions on clock `y` between the edges last_is_1 $\Leftrightarrow\!\!\rightarrow$ next_is_01 and last_is_1 $\Leftrightarrow\!\!\rightarrow$ up_0 in the receiver automaton.

Once again a diagnostic trace was produced, partially shown in Figure 4. This time the error was found by inspection of the action sequence. In the control-location (head_is_0,tranhigh_0a,last_is_1,ack) three `output_1`-actions and one `output_0` have been performed but the value of the variable `m` indicates an odd parity of the accumulated output bit stream. It was concluded that some update operation of `m` was

$$
\begin{array}{l}
\vdots \\
((\mathsf{head\_is\_0}, \mathsf{tranhigh\_0a}, \mathsf{up\_1}, \mathsf{ack}), (0,0), (2,0,1,2)) \xrightarrow{\texttt{output\_1}} \\
((\mathsf{head\_is\_0}, \mathsf{tranhigh\_0a}, \mathsf{last\_is\_1}, \mathsf{ack}), (0,0), (0,0,1,1)) \xrightarrow{\epsilon(76)} \\
((\mathsf{head\_is\_0}, \mathsf{tranhigh\_0a}, \mathsf{last\_is\_1}, \mathsf{ack}), (76,76), (0,0,1,1)) \xrightarrow{\texttt{input\_0}} \\
\vdots
\end{array}
$$

Figure 4: Diagnostic trace from the second improved version of the protocol.

wrong or missing.

5.2.4 The Final Version

When the modulo-2 addition on the variable `m` was removed from the edges $\mathsf{last\_is\_1} \Leftrightarrow \mathsf{next\_is\_01}$ and $\mathsf{last\_is\_0} \Leftrightarrow \mathsf{next\_is\_01}$ in the receiver automaton we got the final version of the protocol, shown in Figure 8. Not only is the correctness property satisfied by this version, but also will the protocol terminate normally in its final state (stop,stop,stop,ack). Furthermore, by adjusting the rate of the clocks (i.e. `x` and `y`), in the sender and the receiver respectively, it can be confirmed that the correctness property is not satisfied if the tolerance is greater or equal to $\pm\frac{1}{17}$.

6 Conclusion and Future Work

In this paper, we have presented a diagnostic model-check algorithm for real-time systems modelled as networks of timed automata. Like other model-checking algorithms for timed automata, the presented algorithm is capable of checking if a logical property is satisfied by an abstract model of an real-time system. However, unlike most other algorithms, our algorithm is specifically designed to support the difficult task of debugging system models whenever the verification of a particular property fails. A diagnostic trace is then automatically generated, which violates the currently checked logical property, but which is a possible trace of the analysed system. Such a violating trace may be considered as diagnostic information of the error, useful during the subsequent debugging of the system model.

The presented algorithm has been implemented in the verification tool UPPAAL. To show how the diagnostic feature of UPPAAL can be used, we have presented a case study where a version of Philips audio-control protocol is analysed. In the case-study we adopted an iterative procedure in which protocol models were successively refined based on the output of earlier verifications. After three iterations the protocol was found to satisfy the main correctness criteria expressing that the bit sequences received on the receiving node always match the sent bit sequences.

Besides a diagnostic model-checker for networks of timed automata, the UPPAAL tool have a graphical interface based on Autograph, allowing system descriptions to be defined by drawing and thereby allowing the user to see what is verified. In this way, a number of errors can be avoided. In a case study, both the graphical interface

and the automatically generated diagnostic traces proved useful for detecting and correcting several errors in the description of the protocol.

Acknowledgements

The UPPAAL tool has been implemented in large parts by Johan Bengtsson and Fredrik Larsson. The authors would like to thank them for their excellent work and also for several discussions concerning the verification of Philips audio-control protocol.

References

[AD90] Rajeev Alur and David Dill. Automata for Modelling Real-Time Systems. In *Proc. of Int. Colloquium on Algorithms, Languages and Programming*, number 443 in Lecture Notes in Computer Science, pages 322–335, July 1990.

[AL92] Martin Abadi and Leslie Lamport. An Old-Fashioned Recipe for Real Time. In *Proc. of REX Workshop "Real-Time: Theory in Practice"*, number 600 in Lecture Notes in Computer Science, 1992.

[BPV94] D. Bosscher, I. Polak, and F. Vaandrager. Verification of an Audio-Control Protocol. In *Proc. of Formal Techniques in Real-Time and Fault-Tolerant Systems*, number 863 in Lecture Notes in Computer Science, 1994.

[Dil89] David Dill. Timing Assumptions and Verification of Finite-State Concurrent Systems. In J. Sifakis, editor, *Proc. of Automatic Verification Methods for Finite State Systems*, number 407 in Lecture Notes in Computer Science, pages 197–212. Springer–Verlag, 1989.

[DY95] C. Daws and S. Yovine. Two examples of verification of multirate timed automata with KRONOS. In *Proc. of the 16*th *IEEE Real-Time Systems Symposium*, pages 66–75. IEEE Computer Society Press, December 1995.

[HL89] Hans Hüttel and Kim G. Larsen. The use of static constructs in a modal process logic. In *Logic at Botik'89*, number 363 in Lecture Notes in Computer Science, pages 163–180. Springer–Verlag, 1989.

[HNSY94] Thomas. A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic Model Checking for Real-Time Systems. *Information and Computation*, 111(2):193–244, 1994.

[HWT95] Pei-Hsin Ho and Howard Wong-Toi. Automated Analysis of an Audio Control Protocol. In *Proc. of the 7*th *Int. Conf. on Computer Aided Verification*, number 939 in Lecture Notes in Computer Science. Springer–Verlag, 1995.

[LLW95] Francois Laroussinie, Kim G. Larsen, and Carsten Weise. From Timed Automata to Logic — and Back. In *Proc. of MFCS'95*, Lecture Notes in Computer Sciencie, 1995. Also BRICS report series RS–95–2.

[LPY95] Kim G. Larsen, Paul Pettersson, and Wang Yi. Compositional and Symbolic Model-Checking of Real-Time Systems. In *Proc. of the 16*th *IEEE Real-Time Systems Symposium*, pages 76–87. IEEE Computer Society Press, December 1995.

[OSY94] A. Olivero, J. Sifakis, and S. Yovine. Using Abstractions for the Verification of Linear Hybrids Systems. In *Proc. of the 6*th *Int. Conf. on Computer Aided Verification*, number 818 in Lecture Notes in Computer Science, 1994.

[Yi91] Wang Yi. *A Calculus of Real Time Systems.* PhD thesis, Department of Computer Science, Chalmers University of Technology, 1991.

[YPD94] Wang Yi, Paul Pettersson, and Mats Daniels. Automatic Verification of Real-Time Communicating Systems By Constraint-Solving. In Dieter Hogrefe and Stefan Leue, editors, *Proc. of the 7*th *Int. Conf. on Formal Description Techniques*, pages 223–238. North–Holland, 1994.

A The System Descriptions

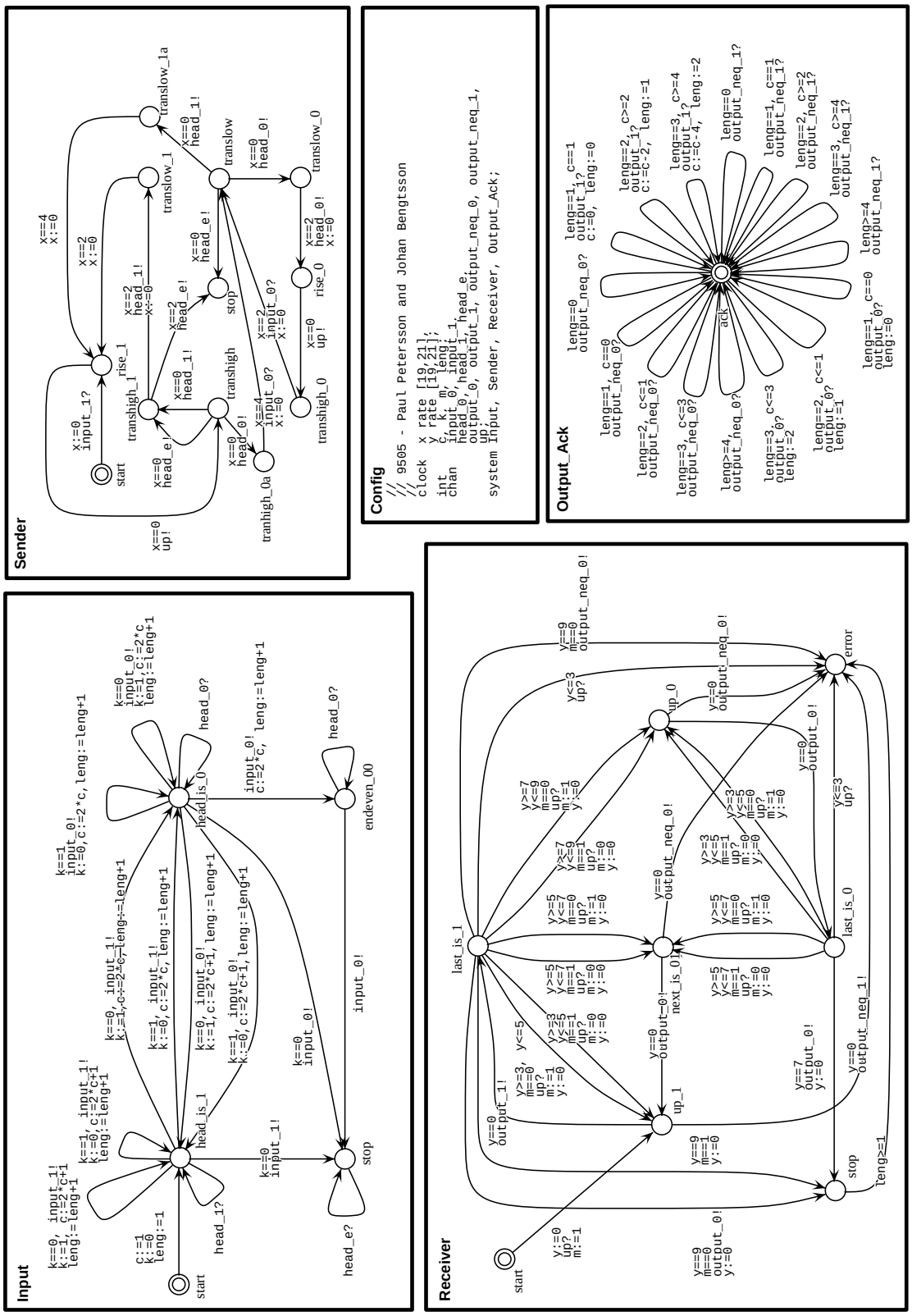


Figure 5: Philips Audio-Control Protocol — First Version.

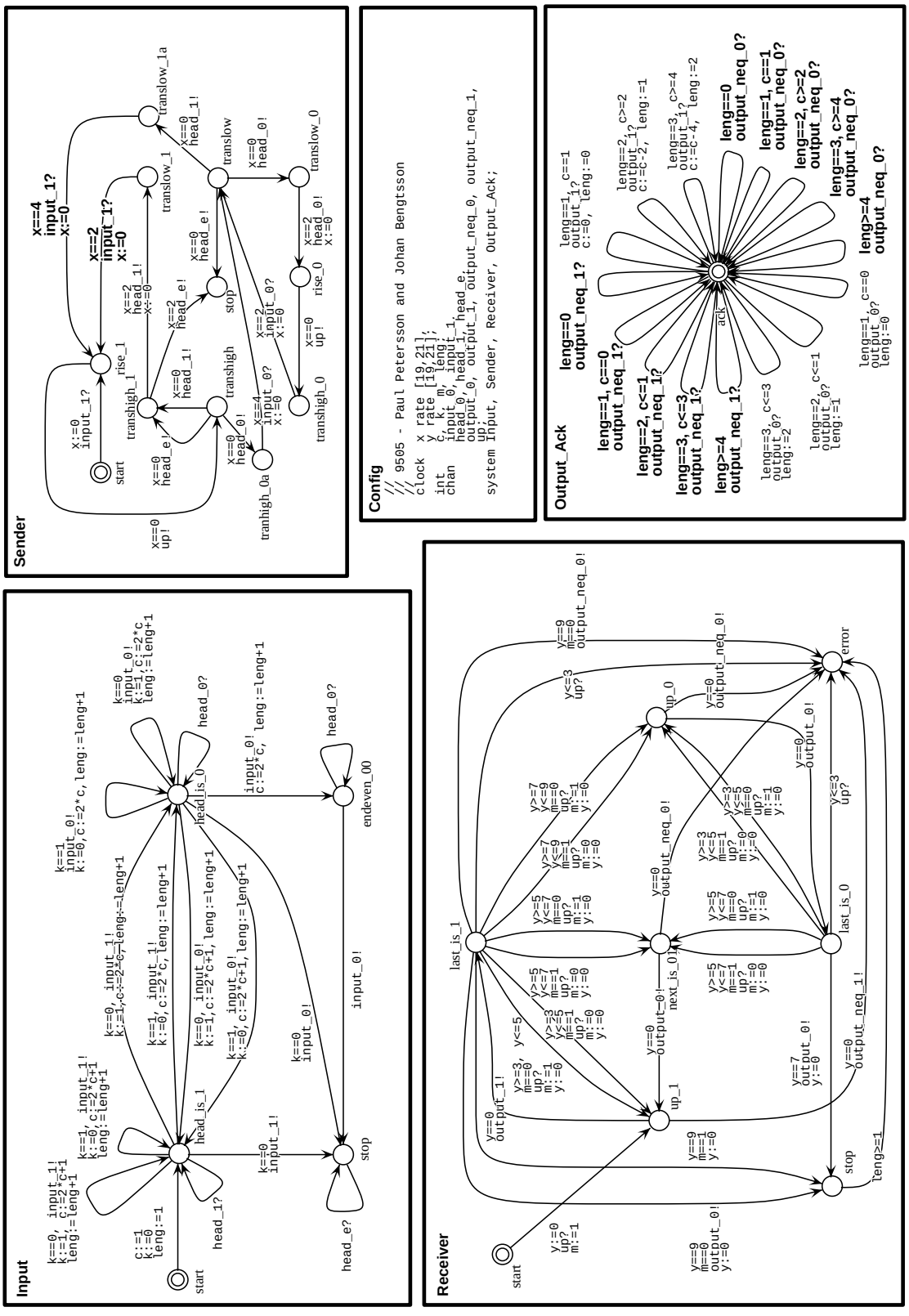


Figure 6: Philips Audio-Control Protocol — First Improved Version.

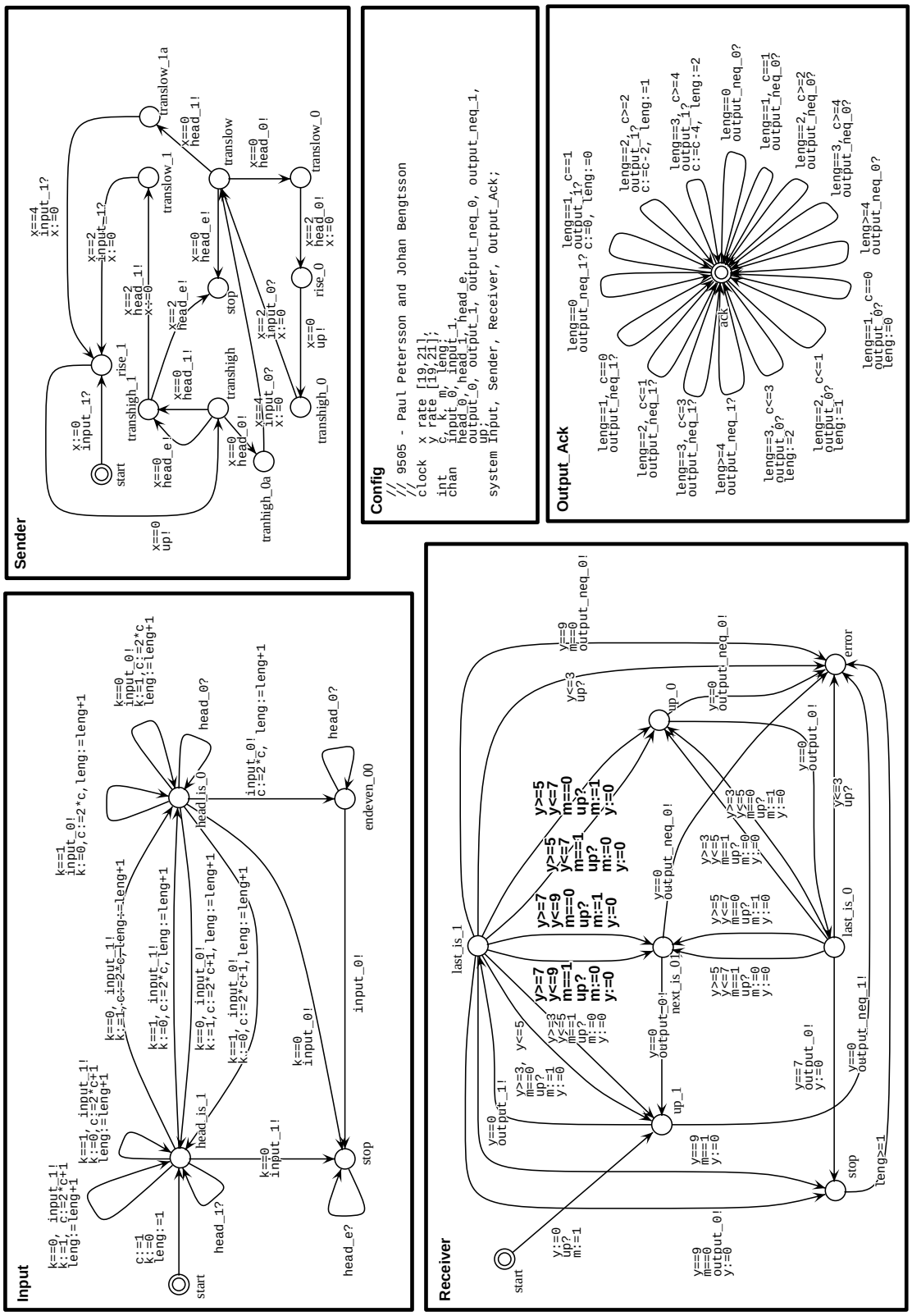


Figure 7: Philips Audio-Control Protocol — Second Improved Version.

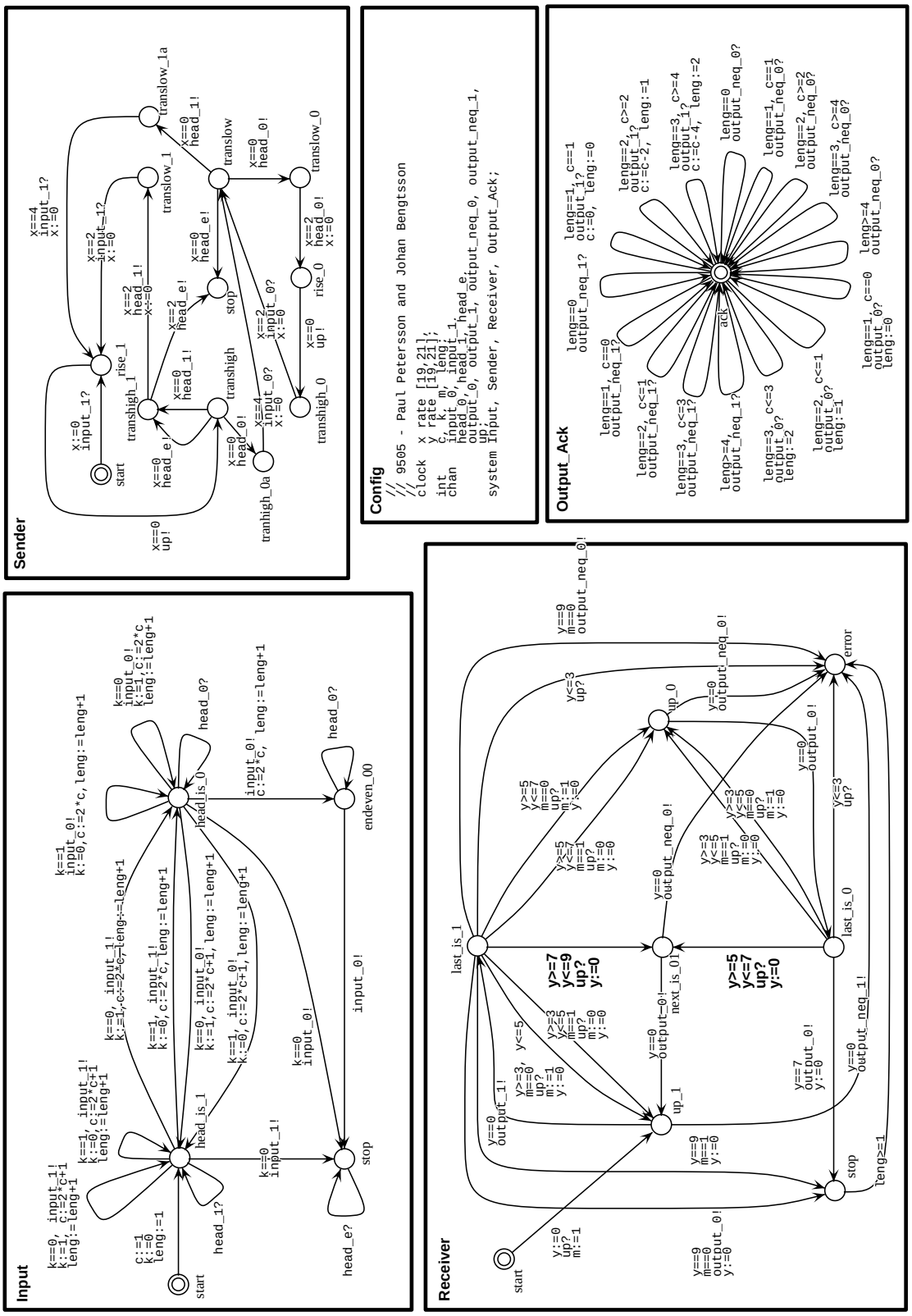


Figure 8: Philips Audio-Control Protocol — Final Version.

B Proof of Theorem 4.1

We only show (1) as the proof of (2) is similar. The proof is by structural induction over φ. Let $Tr(l, D)$ denote all traces of $\mathcal{S}_A$ starting from the states (l, u) with $u \in D{\downarrow}\mathcal{C}$. Assume as induction hypothesise (I.H.) $\Big(\ \forall \sigma \in Tr(l, D)\ :\ \sigma \not\models (l, D) : \varphi \Rightarrow \neg(\sigma \models_v \varphi)$ and $v \in D{\downarrow}K\ \Big)$ for all (l, D). In the proof we shall write $\models$ instead of $\models_v$ and omit $v \in D{\downarrow}K$ whenever it is not confusing. We need to prove one case for each operator in the logic :

($\varphi = p$) Assume $\sigma \not\models (l, D) : p$. Due to Table 2 we have $p \notin V(l),\ w \in D,\ u = w{\downarrow}\mathcal{C}$ and $\sigma = (l, u) \Leftrightarrow\!\!\rightarrow \cdots$. From Table 1 we get $\neg(\sigma \models p)$.

($\varphi = c$) Assume $\sigma \not\models (l, D) : c$. From Table 2 we get $w \in D \wedge \neg c,\ v = w{\downarrow}K,\ u = w{\downarrow}\mathcal{C}$ and $\sigma = (l, u) \Leftrightarrow\!\!\rightarrow \cdots$. It follows that $\neg c(v)$ and further from Table 1 we have $\neg((\sigma \models_v c)$.

($\varphi = \psi_1 \wedge \psi_2$) Assume $\sigma \not\models (l, D) : \psi_1 \wedge \psi_2$. By Table 2 we have $\sigma \not\models (l, D) : \psi_1$ and $\sigma \not\models (l, D) : \psi_2$, which by I.H. implies $\neg((\sigma \models \psi_1)$ and $(\sigma \models \psi_2))$. By Table 1 we have $(\sigma \models \psi_1 \wedge \psi_2)$.

($\varphi = p \vee \psi$) Assume $\sigma \not\models (l, D) : p \vee \psi$. Due to Table 2 we get $\sigma \not\models (l, D) : p$ and $\sigma \not\models (l, D) : \psi$. By I.H. this implies $p \notin V(\sigma)$ and $\sigma \not\models (l, D) : \psi$, which is the same as $\neg(p \in V(\sigma)$ or $\sigma \models \psi)$. Further due to Table 1 we have $\neg(\sigma \models p \vee \psi)$.

($\varphi = c \vee \psi$) Assume $\sigma \not\models (l, D) : c \vee \psi$. By Table 2 we have $\sigma \not\models (l, D \wedge \neg c) : \varphi$, which by I.H. implies $\neg(\sigma \models_v \psi)$ and $v \in (D \wedge \neg c){\downarrow}K$. This is equivalent to $\neg(\sigma \models_v \psi$ or $c(v))$ and $v \in D{\downarrow}K$. It follows from Table 1 that $\neg(\sigma \models_v c \vee \psi)$ and $v \in D{\downarrow}K$.

($\varphi = \mathsf{INV}(\psi)$) Assume $\sigma \not\models (l, D) : \mathsf{INV}(\psi)$. By Table 2 we have $\Big((\sigma \not\models (l, D) : \psi)$ or $(\sigma' \not\models (l, D^{\uparrow l}) : \mathsf{INV}(\psi)$ where $\sigma = (l, u) \overset{\epsilon(d)}{\Leftrightarrow\!\!\rightarrow} \sigma',\ u \in D{\downarrow}\mathcal{C},\ u' = u \oplus d,$ $\sigma' = (l, u') \Leftrightarrow\!\!\rightarrow \ldots$, and $d \in I\!R_+)$ or $(\sigma'' \not\models (l', r(g \wedge D)) : \mathsf{INV}(\psi)$ whenever $l \overset{g\,a\,r}{\Leftrightarrow\!\!\rightarrow} l',\ \sigma = (l, u) \overset{a}{\Leftrightarrow\!\!\rightarrow} \sigma'',\ \sigma'' = (l', u') \Leftrightarrow\!\!\rightarrow \ldots,\ u \in (g \wedge D){\downarrow}\mathcal{C}$ and $u' = r[u]\Big)$. Due to I.H. this implies $\neg\Big((\sigma \models_v \psi$ and $v \in D{\downarrow}K)$ and $(\sigma' \models_{v'} \mathsf{INV}(\psi)$ where $w \in D^{\uparrow l},\ \sigma = (l, u) \overset{\epsilon(d)}{\Leftrightarrow\!\!\rightarrow} \sigma'\ u = w{\downarrow}\mathcal{C},\ u' = u \oplus d,\ \sigma' = (l, u') \Leftrightarrow\!\!\rightarrow \ldots,\ d \in I\!R_+,$ and $v' \in w{\downarrow}K)$ and $(\sigma'' \models_{v''} \mathsf{INV}(\psi)$ whenever $l \overset{g\,a\,r}{\Leftrightarrow\!\!\rightarrow} l',\ \sigma = (l, u) \overset{a}{\Leftrightarrow\!\!\rightarrow} \sigma'',$ $w' \in (g \wedge D)\ u = w'{\downarrow}\mathcal{C},\ \sigma'' = (l', u') \Leftrightarrow\!\!\rightarrow \ldots\ u' = r[u],\ v'' = w'{\downarrow}K)\Big)$. Thus, $\mathsf{INV}(\psi)$ is not satisfied by any position π reachable in one step from (l, u) where $u \in D$. It follows from Table 1 that $\neg(\sigma \models_v \mathsf{INV}(\psi))$ and $v \in D{\downarrow}K$.

($\varphi = \psi \mathsf{UNTIL}_{\emptyset} a$) We prove the case for $a = c$. The case for $a = p$ is similar. Assume $\sigma \not\models (l, D) : \psi \mathsf{UNTIL}_{\emptyset} c$. By Table 2 we have $\Big((\sigma \not\models (l, D \wedge \neg c) : \psi)$ or $(\sigma' \not\models (l, (D \wedge \neg c)^{\uparrow l}) : (\psi \mathsf{UNTIL}_{\emptyset} c)$ where $\sigma = (l, u) \overset{\epsilon(d)}{\Leftrightarrow\!\!\rightarrow} \sigma',\ \sigma' = (l, u') \Leftrightarrow\!\!\rightarrow \ldots,$ $u \in (D \wedge \neg c){\downarrow}\mathcal{C},\ u' = u \oplus d)$ or $(\sigma'' \not\models (l, r(g \wedge D \wedge \neg c)) : (\psi \mathsf{UNTIL}_{\emptyset} c)$ whenever $l \overset{g\,a\,r}{\Leftrightarrow\!\!\rightarrow} l',\ \sigma = (l, u) \overset{a}{\Leftrightarrow\!\!\rightarrow} \sigma'',\ \sigma'' = (l', u') \Leftrightarrow\!\!\rightarrow \ldots,\ u \in (D \wedge g \wedge \neg c){\downarrow}\mathcal{C},\ u' = r[u])\Big)$. By the induction hypothesise this implies $\neg\Big((\sigma \models_v \psi$ and $v \in (D \wedge \neg c){\downarrow}K)$ and $(\sigma' \models_{v'} (\psi \mathsf{UNTIL}_{\emptyset} c)$ where $w' \in (D \wedge \neg c)^{\uparrow l},\ v' = w'{\downarrow}K,\ \sigma = (l, u) \overset{\epsilon(d)}{\Leftrightarrow\!\!\rightarrow} \sigma',\ \sigma' =$

$(l,u') \Leftrightarrow\!\!\rightarrow \ldots$, $u = w'{\downarrow}\mathcal{C}$, $u' = u \oplus d$) and ($\sigma'' \models_{v''} (\psi \mathsf{UNTIL}_{\emptyset} c)$, $w'' \in D \wedge g \wedge \neg c$ and $v'' = w''{\downarrow}K$ whenever $l \overset{g\,a\,r}{\Leftrightarrow\!\!\rightarrow} l'$, $\sigma = (l,u) \overset{a}{\Leftrightarrow\!\!\rightarrow} \sigma''$, $\sigma'' = (l',u') \Leftrightarrow\!\!\rightarrow \ldots$, $u = w''{\downarrow}\mathcal{C}, u' = r[u]\ )\Big)$. Due to Table 1 we have $\neg(\sigma \models_v (\psi \mathsf{UNTIL}_{\emptyset} c))$ and $v = D{\downarrow}K$.

($\varphi = \psi \mathsf{UNTIL}_r a$)) Assume $\sigma \not\models (l,D) : \psi \mathsf{UNTIL}_r a$. We have by Table 2 that $\sigma \not\models (l, r(D)) : \psi \mathsf{UNTIL}_{\emptyset} a$. This implies $\neg(\sigma \models_{r[v]} \psi \mathsf{UNTIL}_{\emptyset} a)$ and $v \in D{\downarrow}K$ due to the induction hypothesis. Further, from Table 1 we have $\neg(\sigma \models_v \psi \mathsf{UNTIL}_r a)$ and $v \in D{\downarrow}K$. □

Paper D

Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. Efficient Verification of Real-Time Systems: Compact Data Structures and State-Space Reduction. In *Proceedings of the 18*th *IEEE Real-Time Systems Symposium*, pages 14–24. IEEE Computer Society Press 1997.

Efficient Verification of Real-Time Systems: Compact Data Structures and State-Space Reduction

Kim G. Larsen*, Fredrik Larsson†, Paul Pettersson†, and Wang Yi†

* BRICS, Aalborg University, Denmark. Email: kgl@cs.auc.dk.

† Department of Computer Systems, Uppsala University, Sweden.
Email: {fredrikl,paupet,yi}docs.uu.se.

Abstract. During the past few years, a number of verification tools have been developed for real-time systems in the framework of timed automata. One of the major problems in applying these tools to industrial-size systems is the huge memory-usage for the exploration of the state-space of a network (or product) of timed automata, as the model-checkers must keep information about not only the control structure of the automata but also the clock values specified by clock constraints.

In this paper, we present a compact data structure for representing clock constraints. The data structure is based on an $\mathcal{O}(n^3)$ algorithm which, given a constraint system over real-valued variables consisting of bounds on differences, constructs an equivalent system with a *minimal* number of constraints. In addition, we have developed an on-the-fly reduction technique to minimise the space-usage. Based on static analysis of the control structure of a network of timed automata, we are able to compute a set of symbolic states that cover all the dynamic loops of the network in an on-the-fly searching algorithm, and thus ensure termination in reachability analysis.

The two techniques and their combination have been implemented in the tool UPPAAL. Our experimental results demonstrate that the techniques result in truly significant space-reductions: for six examples from the literature, the space saving is between 75% and 94%, and in (nearly) all examples time-performance is improved. Noteworthy is also the observation that the two techniques are completely orthogonal.

1 Introduction

Reachability analysis has been one of the most successful methods for automated analysis of concurrent systems. Many verification problems e.g. invariant checking can be solved by means of reachability analysis. It can in many cases also be used for checking whether a system described as an automaton satisfies a requirement specification formulated e.g. in linear temporal logic, by converting the requirement to an automaton and thereafter checking whether the parallel composition of the system and requirement automata can reach certain annotated states [VW86, Hol91, ABL98]. However, the major problem in applying reachability analysis is the potential combinatorial explosion of state spaces. To attack this problem, various symbolic and reduction techniques have been put forward over the last decade to efficiently represent state space and to avoid exhaustive state space exploration (e.g. [BCM+90, GW91, Val90, CFJ93, CGL92, EJ93, And95]); such techniques have played a crucial role for the successful development of verification tools for finite-state systems.

In the last few years, new verification tools have been developed, for the class of

```
PASSED:= {}
WAITING:= {(l_0, D_0)}
repeat
    begin
        get (l, D) from WAITING
        if (l, D) ⊨ φ then return "YES"
        else if D ⊈ D' for all (l, D') ∈ PASSED then
            begin
                add (l, D) to PASSED
                SUCC:={(l_s, D_s) | (l, D) ⇝ (l_s, D_s) and D_s ≠ ∅}
                for all (l_s, D_s) in SUCC do
                    put (l_s, D_s) to WAITING
            end
    end
until WAITING={}
return "NO"
```

Figure 1: An Algorithm for Symbolic Reachability Analysis.

infinite-state systems known as timed systems [HHWT95, DY95, BLL+96]. Notably the verification engines of most tools in this category are based on reachability analysis of timed automata following the pioneering work of Alur and Dill [AD90]. A timed automaton is an extension of a finite automaton with a finite set of real-valued clock-variables. The foundation for decidability of reachability problems for timed automata is Alur and Dill's region technique, by which the infinite state space of a timed automaton due to the density of time, may effectively be partitioned into finitely many equivalence classes i.e. *regions* in such a way that states within each class will always evolve to states within the same classes. However, reachability analysis based on the region technique is practically infeasible due to the potential state explosions arising from not only the control-structure (as for finite-state systems) but also the region space [LPY95a].

Efficient data structures and algorithms have been sought to represent and manipulate timing constraints over clock variables (e.g. by Difference Bounded Matrices [Bel57, Dil89], or Binary Decision Diagrams [BCM+90, AMP97]) and to avoid exhaustive state space exploration (e.g. by application of partial order reductions [GW91, Val90, Pag96] or compositional methods [And95, LPY95a]). One of the main achievements in these studies is the symbolic technique [Dil89, YL93, HNSY94, YPD94, LPY95a], that converts the reachability problem to that of solving simple constraints. The technique can be simply formulated in an abstract reachability algorithm[1] as shown in Figure 1. The algorithm is to check whether a timed automaton may reach a state satisfying a given state formula φ. It explores the state space of the automaton in terms of *symbolic states* in the form (l, D) where l is a control-node

[1]Several verification tools for timed systems (e.g. UPPAAL [BLL+96]) have been implemented based on this algorithm.

and D is a constraint over clocks variables.

We observe that several operations of the algorithm are critical for efficient implementations. First, the algorithm depends heavily on the test operations for checking the inclusion $D \subseteq D'$ (i.e. the inclusion between the solution sets of D, D') and the emptiness of D_s in constructing the successor set SUCC of (l, D). Clearly, it is important to design efficient data structures and algorithms for the representation and manipulation of clock constraints. One such well-known data structure is that of DBM (*Difference Bounded Matrix*), which offers a canonical representation for constraint systems. It has been successfully used in several real-time verification tools, e.g. UPPAAL [BLL+96] and KRONOS [DY95]. A DBM representation is in fact a weighted directed graph where the vertices correspond to clocks (including a zero-clock) and the weights on the edges stand for the bounds on the differences between pairs of clocks [Bel57, Dil89, YL93]. As it gives an explicit bound for the difference between each pair of clocks, its space-usage is in the order of $\mathcal{O}(n^2)$ where n is the number of clocks. However, in practice it often turns out that most of these bounds are redundant.

In this paper, we present a compact data structure for DBM, which provides *minimal* and *canonical* representations of clock constraints and also allows for efficient inclusion checks. We have developed an $\mathcal{O}(n^3)$ algorithm that given a DBM constructs a minimal number of constraints equivalent to the original constraints represented by the DBM (i.e. with the same solution set). The algorithm is essentially a minimisation algorithm for weighted directed graphs, and hence solves a problem of independent interest. Note that the main global data structure of the algorithm in Figure 1 is the passed list (i.e. PASSED) holding the explored states. In many cases, it will store all the reachable symbolic states of the automaton. Thus, it is desirable that when saving a (symbolic) state in the passed list, we save the (often substantially smaller) minimal constraint system. The minimal representation also makes the inclusion-checking of the algorithm more efficient. Our experimental results demonstrate truly significant space-savings as well as better time-performance (see statistics in section 5).

In addition to the *local* reduction technique above, which is to minimise the space-usage of each individual symbolic state, as the second contribution of this paper, we have developed a *global* reduction technique to reduce the total number of states to save in the global data structure, i.e. the passed list. It is completely orthogonal to the local technique. In the abstract algorithm of Figure 1, we notice the step of saving the new encountered state (l, D) in the passed list when the inclusion-checking for $D \subseteq D'$ fails (i.e. $D \not\subseteq D'$). Its purpose is first of all to guarantee termination but also to avoid repeated exploration of states that have several predecessors. However, this is not necessary if all the predecessors of (l, D) are already present in the passed list. In fact, to ensure termination, it suffices to save only one state for each dynamic loop. An improved on-the-fly reachability algorithm according to the global reduction strategy has been implemented in UPPAAL [BLL+96] based on static analysis of the control structure of timed automata. Our experimental results demonstrate significant space-savings and also better time-performance (see statistics in section 5).

The outline of this paper is as follows: In the next section we review the seman-

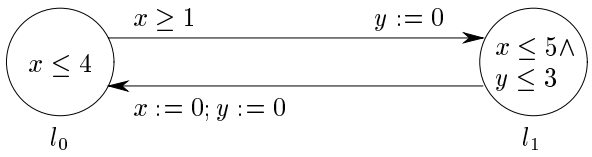


Figure 2: A Timed Automaton.

tics of timed automata and the notion of Difference Bounded Matrix (DBM) for clock constraints. Section 3 presents the compact data structure for DBM and the local reduction technique (i.e. the minimisation algorithm for weighted directed graphs). Section 4 is devoted to develop the global reduction technique based on control structure analysis. Section 5 presents our experimental results for both techniques and their combination. Section 6 concludes the paper.

2 Preliminaries

2.1 Timed Automata

The model of timed automata was first introduced in [AD90] and has since then established itself as a standard model for real-time systems. For the reader not familiar with the notion of timed automata we give a short informal description.

Consider the timed automaton of Figure 2. It has two control nodes l_0 and l_1 and two real-valued clocks x and y. A *state* of the automaton is of the form (l, s, t), where l is a control node, and s and t are non-negative reals giving the value of the two clocks x and y. A control node is labelled with a condition (the invariant) on the clock values that must be satisfied for states involving this node. Assuming that the automaton starts to operate in the state $(l_0, 0, 0)$, it may stay in node l_0 as long as the invariant $x \leq 4$ of l_0 is satisfied. During this time the values of the clocks increase synchronously. Thus from the initial state, all states of the form (l_0, t, t), where $t \leq 4$, are reachable. The edges of a timed automaton may be decorated with a condition (guard) on the clock values that must be satisfied in order to be enabled. Thus, only for the states (l_0, t, t), where $1 \leq t \leq 4$, is the edge from l_0 to l_1 enabled. Additionally, edges may be labelled with simple assignments reseting clocks. For example, when following the edge from l_0 to l_1 the clock y is reset to 0 leading to states of the form $(l_1, t, 0)$, where $1 \leq t \leq 4$.

In general, a timed automaton is a standard finite-state automaton extended with a finite collection $\mathcal{C}$ of real-valued clocks ranged over by x, y etc. We use $\mathcal{B}(\mathcal{C})$ ranged over by g (and latter D), to stand for the set of formulas that can be an atomic constraint of the form: $x \sim n$ or $x \Leftrightarrow y \sim n$ for $x, y \in \mathcal{C}$, $\sim \in \{\leq, \geq\}$[2] and n being a natural number, or a conjunction of such formulas. Elements of $\mathcal{B}(\mathcal{C})$ are called *clock constraints* or *constraint systems* over $\mathcal{C}$.

[2]For reasons of simplicity and clarity in presentation we have chosen only to consider the non-strict orderings. However, the techniques given extends easily to strict orderings.

Definition 2.1 (Timed Automata) *A timed automaton A over clocks $\mathcal{C}$ is a tuple $\langle N, l_0, \Leftrightarrow\!\!\rightarrow, I\rangle$ where N is a finite set of nodes (control-nodes), l_0 is the initial node, $\Leftrightarrow\!\!\rightarrow \subseteq N \times \mathcal{B}(\mathcal{C}) \times 2^{\mathcal{C}} \times N$ corresponds to the set of edges, and finally, $I : N \mapsto \mathcal{B}(\mathcal{C})$ assigns invariants to nodes. In the case, $\langle l, g, r, l'\rangle \in \Leftrightarrow\!\!\rightarrow$, we write $l \overset{g,r}{\Leftrightarrow\!\!\rightarrow} l'$.* □

Formally, we represent the values of clocks as functions (called clock assignments) from $\mathcal{C}$ to the non-negative reals $\mathbb{R}_+$. We denote by $\mathbb{R}_+^{\mathcal{C}}$ the set of clock assignments for $\mathcal{C}$. A semantical *state* of an automaton A is now a pair (l, u), where l is a node of A and u is a clock assignment for $\mathcal{C}$, and the semantics of A is given by a transition system with the following two types of transitions (corresponding to delay-transitions and edge-transitions):

- $(l, u) \Leftrightarrow\!\!\rightarrow (l, u \oplus d)$ if $I(l)(u)$ and $I(l)(u \oplus d)$
- $(l, u) \Leftrightarrow\!\!\rightarrow (l', u')$ if there exist g and r such that $l \overset{g,r}{\Leftrightarrow\!\!\rightarrow} l'$, $g(u)$ and $u' = r[u]$

where for $d \in \mathbb{R}_+$, $u \oplus d$ denotes the time assignment which maps each clock x in $\mathcal{C}$ to the value $u(x) + d$, and for $r \subseteq \mathcal{C}$, $r[u]$ denotes the assignment for $\mathcal{C}$ which maps each clock in r to the value 0 and agrees with u over $\mathcal{C} \backslash r$.

Clearly, the semantics of a timed automaton yields an infinite transition system, and is thus not an appropriate basis for decision algorithms. However, efficient algorithms may be obtained using a finite-state *symbolic* semantics based on *symbolic states* of the form (l, D), where $D \in \mathcal{B}(\mathcal{C})$ [HNSY94, YPD94]. We shall consider a clock constraint as a set of clock assignments and use $u \in D$ to stand for u satisfied D.

The symbolic counterpart to the standard semantics is given by the following two (fairly obvious) types of symbolic transitions:

- $(l, D) \rightsquigarrow \Big(l, (D \wedge I(l))^{\uparrow} \wedge I(l)\Big)$
- $(l, D) \rightsquigarrow \Big(l', r(g \wedge D)\Big)$ if $l \overset{g,r}{\Leftrightarrow\!\!\rightarrow} l'$

where $D^{\uparrow} = \{u \oplus d \mid u \in D \wedge d \in \mathbb{R}_+\}$ and $r(D) = \{r[u] \mid u \in D\}$. It may be shown that $\mathcal{B}(\mathcal{C})$ (the set of clock constraints) is closed under these two operations (and $\wedge$) [Dil89]. Moreover, the symbolic semantics characterise the standard semantics in the sense that, whenever $u \in D$ and $(l, D) \rightsquigarrow (l', D')$ then $(l, u) \Leftrightarrow\!\!\rightarrow (l', u')$ for $u' \in D'$.

Finally, we introduce the notion of networks of timed automata [YPD94, LPY95a]. A network is the parallel composition of a finite set of automata for a given synchronisation function. To illustrate the on-the-fly verification technique, we only need to study the case dealing with interleaving, that is, the network of automata $A_1 \ldots A_n$, is the Cartesian product of A_i's. Assume a vector l of control nodes. We shall use $l[i]$ to stand for the ith element of l and $l[l_i'/l_i]$ for the vector where the ith element l_i of l is replaced by l_i'. A control node (i.e. *control vector*) l of a network $A_1 \ldots A_n$ is a vector where $l[i]$ is a node of A_i and the invariant $I(l)$ of l is the conjunction of $I(l[1]) \ldots I(l[n])$. The symbolic semantics of networks is given in terms of control vectors [LPY95a].

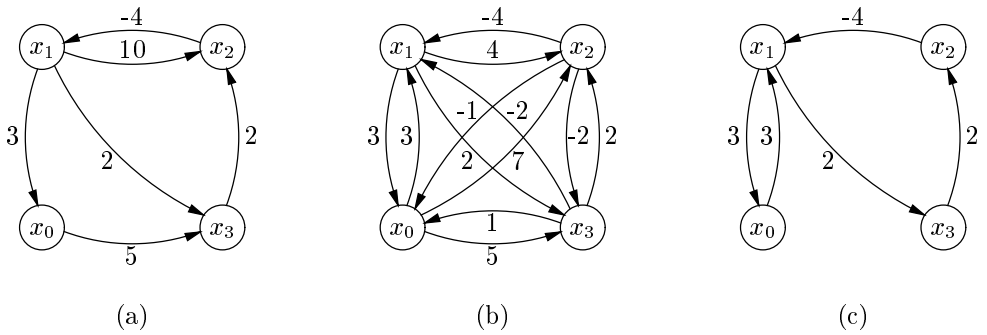


Figure 3: Graph for E (a), its shortest-path closure (b), and shortest-path reduction (c).

- $(l, D) \rightsquigarrow \Big(l, (D \wedge I(l))^{\uparrow} \wedge I(l)\Big)$
- $(l, D) \rightsquigarrow \Big(l[l'_i/l_i], r(g \wedge D)\Big)$ if $l_i \overset{g,r}{\Leftrightarrow} l'_i$

In the later case, we shall say that the symbolic transition is derived by the edge $l_i \overset{g,r}{\Leftrightarrow} l'_i$.

2.2 Difference Bounded Matrices & Shortest-Path Closure

To utilise the symbolic semantics of (networks of) timed automata algorithmically, as for example in the reachability algorithm of Figure 1, it is important to design efficient data structures and algorithms for the representation and manipulation of clock constraints.

One such well-known data structure is that of difference bounded matrices (DBM, see [Bel57, Dil89]), which offers a canonical representation for constraint systems. A DBM representation of a constraint system D is simply a weighted, directed graph, where the vertices correspond to the clocks of C and an additional zero-vertex 0. The graph has an edge from x to y with weight m provided $y \Leftrightarrow x \leq m$ is a constraint of D. Similarly, there is an edge from 0 to x with weight m, whenever $x \leq m$ is a constraint of D [3]. As an example, consider the constraint system E over $\{x_0, x_1, x_2, x_3\}$ being a conjunction of the atomic constraints $x_0 \Leftrightarrow x_1 \leq 3$, $x_3 \Leftrightarrow x_0 \leq 5$, $x_3 \Leftrightarrow x_1 \leq 2$, $x_2 \Leftrightarrow x_3 \leq 2$, $x_2 \Leftrightarrow x_1 \leq 10$, and $x_1 \Leftrightarrow x_2 \leq \Leftrightarrow 4$. The graph representing E is given in Figure 3 (a).

In general, the same set of clock assignments may be described by several constraint systems (and hence graphs). To test for inclusion between constraint systems D and D' [4], which we recall is essential for the termination of the reachability algorithm of Figure 1, it is advantageous if D is *closed under entailment* in the sense that no constraint of D can be strengthened without reducing the solution set. In partic-

[3] We assume that D has been simplified to contain at most one upper and lower bound for each clock and clock-difference.

[4] To be precise, it is the inclusion between the *solution sets* for D and D'.

ular, for D a closed constraint system, $D \subseteq D'$ holds if and only if for any constraint in D' there is a constraint in D at least as tight; i.e. whenever $(x \Leftrightarrow y \leq m') \in D'$ then $(x \Leftrightarrow y \leq m) \in D$ for some $m \leq m'$. Thus, closedness provides a canonical representation, as two closed constraint systems describe the same solution set precisely when they are identical. To close a constraint system D amounts to derive the shortest-path closure for its graph and can thus be computed in time $\mathcal{O}(n^3)$, where n is the number of clocks of D. The graph representation of the closure of the constraint system E from Figure 3 (a) is given in Figure 3 (b). The emptiness-check of a constraint system D simply amounts to checking for negative-weight cycles in its graph representation. Finally, given a closed constraint system D the operations $D^{\uparrow}$ and $r(D)$ may be performed in time $\mathcal{O}(n)$.

3 Minimal Constraint Systems & Shortest Path Reductions

For the reasons stated above a matrix representation of constraint systems in closed form is an attractive data structure, which has been successfully employed by a number of real-time verification tools, e.g. UPPAAL [BLL+96] and KRONOS [DY95]. As it gives an explicit (tightest) bound for the difference between each pair of clocks (and each individual clock), its space-usage is of the order $\mathcal{O}(n^2)$. However, in practice it often turns out that most of these bounds are redundant, and the reachability algorithm of Figure 1 is consequently hampered in two ways by this representation. First, the main data structure PASSED, will in many cases store all the reachable symbolic states of the automaton. Thus, it is desirable, that when saving a symbolic state in the PASSED-list, we save a representation of the constraint system with as few constraints as possible. Secondly, a constraint system D added to the PASSED-list is subsequently only used in checking inclusions of the form $D' \subseteq D$. Recalling the method for inclusion-check from the previous section, we note that (given D' is closed) the time-complexity of the inclusion-check is linear in the number of constraints of D. Thus, again it is advantageous for D to have as few constraints as possible.

In the following subsections we shall present an $\mathcal{O}(n^3)$ algorithm, which given a constraint system constructs an equivalent reduced system with the minimal number of constraints. The reduced constraint system is canonical in the sense that two constrain systems with the same solution set give rise to identical reduced systems. The algorithm is essentially a minimisation algorithm for weighted directed graphs. Given a weighted, directed graph with n vertices, it constructs in time $\mathcal{O}(n^3)$ a reduced graph with the minimal number of edges having the same shortest path closure as the original graph. Figure 3 (c) shows the minimal graph of the graphs in Figure 3 (a) and (b), which is computed by the algorithm.

3.1 Reduction of Zero-Cycle Free Graphs

A weighted, directed graph G is a structure (V, E_G), where V is a finite set of vertices and E_G, is a partial function from $V \times V$ to Z (the integers). The domain of E_G

constitutes the edges of G, and when defined, $E_G(x,y)$ gives the weight of the edge between x and y. We assume that $E_G(x,x) = 0$ for all vertices x, and that G has no cycles with negative weight[5].

Given a graph G, we denote by G^C the *shortest-path closure* of G, i.e. $E_{G^C}(x,y)$ is the length of the shortest path from x to y in G. A *shortest-path reduction* of a graph G is a graph G^R with the minimal number of edges such that $(G^R)^C = G^C$.

The key to reduce a graph is obviously to remove *redundant edges*, where an edge (x,y) is redundant if there exist an alternative path from x to y whose (accumulated) weight does not exceed the weight of the edge itself. E.g. in the graph of Figure 3 (a), the edge (x_1,x_2) is clearly redundant as the accumulated weight of path $(x_1,x_0),(x_0,x_3),(x_3,x_2)$ has a weight (10) not exceeding the weight of the edge itself (also 10). The path $(x_1,x_3),(x_3,x_2)$ makes also the edge (x_1,x_2) redundant. Being redundant, the edge (x_1,x_2) may be removed without changing the shortest-path closure.

Now, consider the edge (x_1,x_2) in the graph of Figure 3 (b). Clearly, the edge is redundant as the path $(x_1,x_3),(x_3,x_2)$ has equal weight. Similarly, the edge (x_3,x_2) is redundant as the path $(x_3,x_1),(x_1,x_2)$ has equal weight. However, though redundant, we cannot just remove the two edges (x_1,x_2) and (x_3,x_2) as removal of one clearly requires the presence of the other. In fact, all edges between the vertices x_1,x_2 and x_3 are redundant, but obviously we cannot remove them all simultaneously. The key explanation of this complicating phenomena is that x_1,x_2,x_3 constitutes a cycle with length zero (a *zero-cycle*). However, for zero-cycle free graphs the situation is the simplest possible:

Lemma 3.1 *Let G_1 and G_2 be zero-cycle free graphs such that ${G_1}^C = {G_2}^C$. If there is an edge $(x,y) \in G_1$ such that $(x,y) \notin G_2$, then $(G_1 \backslash \{(x,y)\})^C = {G_1}^C = {G_2}^C$.*

PROOF: Let α denote the edge (x,y) and let m be the weight of α in G_1. We will show that there is an alternative path in G_1 *not* using α with weight no more than m. From this fact the Lemma obviously follows.

As ${G_1}^C = {G_2}^C$, the shortest path from x to y in G_2 has weight no more than m. As $\alpha \notin G_2$, this path must visit some vertex z different from x and y. Now let m_1 be the shortest path-weight from x to z and let m_2 be the shortest path-weight from z to y; note that G_1 and G_2 agrees on m_1 and m_2, as they have the same shortest-path closure. Then clearly, $m \geq m_1 + m_2$.

Now assume that the shortest path in G_1 from x to z uses $\alpha = (x,y)$. Then, as a sub-path, G_1 will be a path from y to z. Since G_1 also has a path from z to y, it follows that G_1 will have a cycle from y via z back to y. The weight of this cycle can be argued to be no more than $(m_1 \Leftrightarrow m) + m_2$. However, as $m \geq m_1 + m_2$ and there are no negative cycles, this cycle must have weight 0 contradicting the assumption that G_1 is zero-cycle free.

Similarly, a contradiction with the zero-cycle free assumption of G_1 is obtained, if the shortest path in G_1 from z to y uses α. thus we can conclude that there is an path from x to y not using α with length no greater than m. □

[5]This would correspond to constraint systems with empty solution set.

From the above Lemma it follows immediately that all redundant edges of a zero-cycle free graph may be removed without affecting the closure. On the other hand, removal of an edge which is not redundant will of course change the closure of the graph, and must be present in any graph with the same closure. Thus the following theorem follows:

Theorem 3.1 *Let G be a zero-cycle free graph, and let $\{\alpha_1, \ldots, \alpha_2\}$ be the set of redundant edges of G. Then $G^R = G^C \backslash \{\alpha_1, \ldots, \alpha_k\}$.*

PROOF: Follows from Lemma 3.1. □

From an algorithmic point of view, redundancy of edges is easily determined given the closure G^C of a graph G as only path of length 2 needs to be considered: An edge (x, y) is redundant precisely when there is a vertex z $(\neq x, y)$ such that $E_{G^C}(x, y) \geq E_{G^C}(x, z) + E_{G^C}(z, y)$. Thus for zero-cycle free graphs computing G^R is $\mathcal{O}(n^3)$.

3.2 Reduction of Negative-Cycle Free Graphs

For general graphs (without negative cycles) our reduction construct relies on a partitioning of the vertices according to zero-cycles. We say that two vertices x and y are *equivalent* or *zero-equivalent*, if there is a zero-cycle containing them both. We write $x \equiv y$ in this case. Given the closure G^C of a graph G, it is extremely easy to check for zero-equivalence: $x \equiv y$ holds precisely when $E_{G^C}(x, y) = \Leftrightarrow E_{G^C}(y, x)$. Thus, in the graphs of Figure 3 (a) and (b), $\equiv$ partitions the vertices into the two classes $\{x_0\}$ and $\{x_1, x_2, x_3\}$.

To obtain a canonical reduction, we assume that the vertices of G are ordered by assigning them indices as $x_1, x_2, \ldots, x_n$. The equivalence $\equiv$ now induces a natural transformation $G_\equiv$ on the graph G:

Definition 3.1 *Given a graph G, the vertices of the graph $G_\equiv$ are $\equiv$-equivalence classes, denoted E_k, of G. There is an edge between the classes E_i and E_j $(i \neq j)$ if for some $x \in E_i$ and $y \in E_j$ there is an edge in G between x and y. The weight of this edge is $E_{G^C}(E_i^{min}, E_j^{min})$, where E^{min} is the vertex in E with the smallest index.* □

Thus, the distance between E_i and E_j in $G_\equiv$ is the weight of the shortest path in G between the elements of E_i and E_j with smallest index. It is obvious that $G_\equiv$ is a zero-cycle free graph. It is also easy to see that $G_{1\equiv} = G_{2\equiv}$ if ${G_1}^C = {G_2}^C$. Let H be the graph of Figure 3 (a). Then $H_\equiv$ will have vertices $E_0 = \{x_0\}$ and $E_1 = \{x_1, x_2, x_3\}$. The two vertices are connected by two edges both having weight 3.

The following provides a dual to the operator of Definition 3.1:

Definition 3.2 *Let F be a graph with vertices being $\equiv$-equivalence classes with respect to a graph $G = (V, E_G)$. Then the* expansion *of F is a graph F^+ with vertices V and with weight satisfying:*

- *For any multi-member equivalence class* $\{z_1 < z_2 < \cdots < z_k\}$[6] *of* F, F^+ *contains a single cycle* $z_1, z_2, \ldots, z_k, z_1$, *with the weight of the edge* (z_i, z_{i+1}) *being the weight of the shortest path from* z_i *to* z_{i+1} *in* G.
- *Whenever* (E_i, E_j) *is an edge in* F *with weight* m, *then* F^+ *will have an edge from* E_i^{min} *to* E_j^{min} *with weight* m. □

We are now ready to state the main Theorem giving the shortest-path reduction construct for arbitrary negative-cycle free graphs:

Theorem 3.2 *Let* G *be negative-cycle free graph. Then the shortest-path reduction* G^R *of* G *is given by the graph* $({G_\equiv}^R)^+$, *i.e.* $G^R = ({G_\equiv}^R)^+$.

PROOF: We show: (1) that $({G_\equiv}^R)^+$ is a candidate for a shortest-path reduction of G in the sense that $({G_\equiv}^R)^+ = G^C$, and (2) that $({G_\equiv}^R)^+$ is minimal.

1. We first prove that $({G_\equiv}^R)^+ = G^C$. As all edges (x, y) of $({G_\equiv}^R)^+$ have weight of the form $E_{G^C}(x, y)$, it follows that for any path in $({G_\equiv}^R)^+$ there is a path in G with same weight.

   Now consider an edge (x, y) of G. We will demonstrate that there is a path in $({G_\equiv}^R)^+$ with no greater weight.

   - If $x = E_i^{min}$ and $y = E_j^{min}$ for two $\equiv$-classes E_i and E_j, it follows that $E_{G_\equiv}(E_i, E_j) \leq E_G(x, y)$. Furthermore, due to the property of reduction construction, there is a path in ${G_\equiv}^R$ between E_i and E_j with weight no greater than $E_{G_\equiv}(E_i, E_j)$. The same path, but now between the nodes with the minimal indices of the $\equiv$-classes, can be found in $({G_\equiv}^R)^+$. Thus, there is a path in $({G_\equiv}^R)^+$ with weight no greater than $E_G(x, y)$.
   - If $x, y \in E_i$ for some $\equiv$-class E_i, an easy argument gives that $E_{(G_\equiv{}^R)^+}(x, y) = E_{G^C}(x, y) \leq E_G(x, y)$.
   - Consider the case when $x \in E_i$ and $y \in E_j$ for two different $\equiv$-classes, and assume that $E_G(x, y) = m$.
     Now let $m_1 = E_{(G_\equiv{}^R)^+}(x, E_i^{min})$, $m_2 = E_{(G_\equiv{}^R)^+}(E_i^{min}, E_j^{min})$, and $m_3 = E_{(G_\equiv{}^R)^+}(E_j^{min}, y)$. Note that by the reduction construction $m_2 \leq E_{G^C}(E_i^{min}, E_j^{min})$.
     Then there is a path in $({G_\equiv}^R)^+$ from x to y via E_i^{min} and E_j^{min} with weight $m_1 + m_2 + m_3$. Now, if $m < m_1 + m_2 + m_3$, there is a path in G from E_i^{min} to E_j^{min} of weight $m \Leftrightarrow m_1 \Leftrightarrow m_3 < m_2$ contradicting that m_2 is the weight of the shortest path in G between E_i^{min} and E_j^{min}. Thus the path x, E_i^{min}, E_j^{min}, y in $({G_\equiv}^R)^+$ has weight no greater than the edge (x, y) in G.

2. Next we prove that $({G_\equiv}^R)^+$ has minimal number of edges by showing that whenever $H^C = G^C$ then H has at least as many edges as $({G_\equiv}^R)^+$.

   As $H^C = G^C$, H and G induces the same $\equiv$-equivalence relation on the same zero-length cycles. Obviously the fewest edges that will identify k (> 1) vertices,

[6] "$<$" refers to the assumed ordering on the vertices of G.

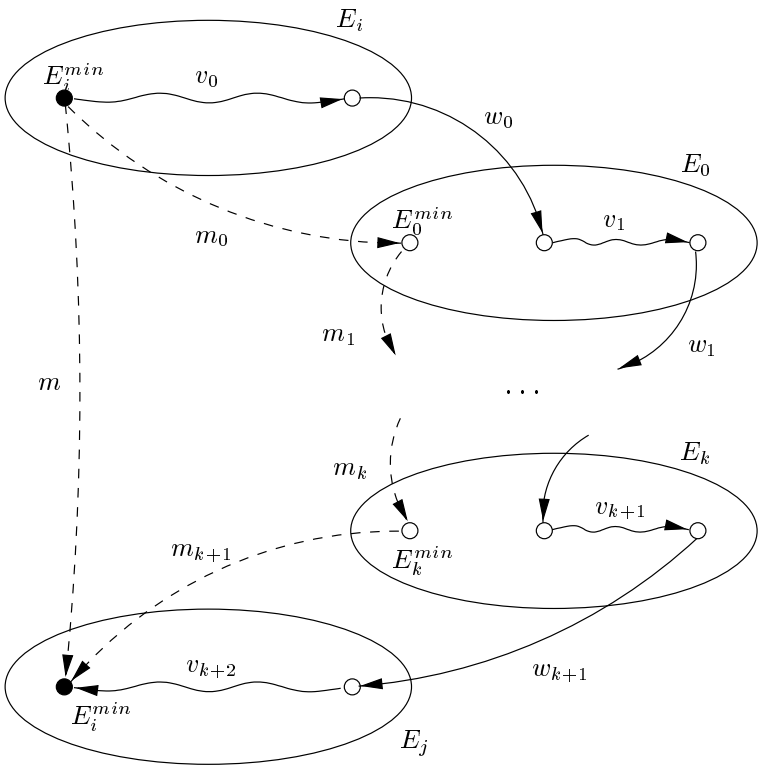


Figure 4: A path in the graph H.

with respect to $\equiv$ is k. Hence, $(G_{\equiv^R})^+$ uses a minimal number of edges between vertices in the same $\equiv$-equivalence class.

Now let (E_i^{min}, E_j^{min}) be an edge in $(G_{\equiv^R})^+$ with weight m. We claim that H must have at least one edge from E_i to E_j.

Assume that this is not the case. Then, as $H^C = G^C$, there must be a path in H from E_i^{min} to E_j^{min} as shown in Figure 4 such that $m = \sum_{i=0}^{k+2} v_i + \sum_{i=0}^{k+1} w_i$.

Now let $m_0 = E_{(G_{\equiv^R})^+}(E_i^{min}, E_0^{min})$, $m_1 = E_{(G_{\equiv^R})^+}(E_0^{min}, E_1^{min})$, $\ldots$, $m_{k+1} = E_{(G_{\equiv^R})^+}(E_k^{min}, E_i^{min})$ (illustrated with dashed lines in Figure 4). Then $m_0 \leq v_0 + w_0 + v_1'$, $m_1 \leq v_1'' + w_1 + v_2'$, $\ldots$, $m_{k+1} \leq v_{k+1}'' + w_{k+1} + v_{k+2}$, where $v_1 = v_1' + v_1''$, $v_2 = v_2' + v_2''$, $\ldots$, $v_{k+1} = v_{k+1}' + v_{k+1}''$.

It follows that $\sum_{i=0}^{k+1} m_i \leq m$. Hence (E_i, E_j) is redundant in $(G_{\equiv^R})^+$ and can be removed, contradicting Lemma 3.1. □

First, note that the above construction of $(G_{\equiv^R})^+$ is well-defined as $G_{\equiv}$ is a zero-cycle free graph and the reduction construction of Theorem 3.1 thus applies. Given the closure G^C of G the constructions of Definitions 3.1 and 3.2 can be computed in $\mathcal{O}(n^2)$. Since G^R is computed from G in $\mathcal{O}(n^3)$, it follows that also $(G_{\equiv^R})^+$ can be constructed in $\mathcal{O}(n^3)$. Now applying the above construction to the graph H of Figure 3 (a), we first note that $H_{\equiv^R} = H_{\equiv}$ as $H_{\equiv}$ has no redundant edges. Expanding $H_{\equiv}$ with respect to the vertex ordering $x_0 < x_1 < x_2 < x_3$ gives the graph of Figure 3 (c), which according to Theorem 3.2 above is the shortest-path reduction of H.

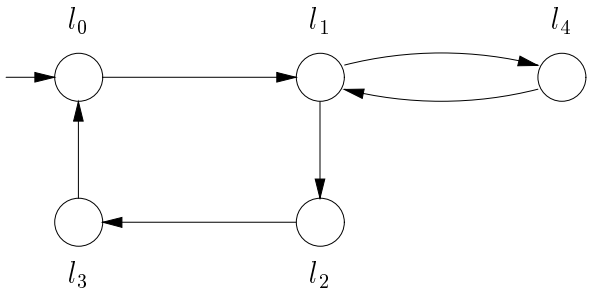


Figure 5: Illustration of Space-Reduction.

Experimental results show that the use of minimal constrain systems (obtained by the above shortest-path reduction algorithm) as a compact data structure leads to truly significant space-savings in practical reachability analysis of timed systems: the space-savings are in the range 68–85%. We refer to Section 5 for more details.

4 Global Reductions and Control Structure Analysis

The preceding section is about *local* reductions in reachability analysis in the sense that the technique developed is for *each* individual symbolic state. In this section, we shall develop a *global* reduction technique to reduce the total number of symbolic states to save in the *global* data structure i.e. the passed list.

4.1 Potential Space-Reductions

We recall the standard reachability analysis algorithm for finite graphs (see e.g. [Pap94]). It is similar to the one in Figure 1, but simpler as no constraints but only control nodes are involved. The algorithm repeats three main operations: *examining* every new encountered node (to see if it is in the passed list), *exploring* the new encountered nodes (computing all their successors for further analysis), and *saving* the explored nodes in the passed list until all reachable nodes are present in the list (i.e. all new encountered nodes are already in the passed list).

Note that the saving of an explored node is to ensure termination and also to avoid repeated exploration of nodes with more than one incoming edge. However it is not necessary to save all reachable nodes. Consider for example, the simple graph in Figure 5 with initial node l_0. Clearly, there is no need to save node l_2, l_3 or l_4 as they will be visited only once if l_1 is present in the passed list.

In fact, to guarantee termination on a finite graph, it is sufficient to save only one node for each cycle in the graph. For example, as l_1 covers the two cycles of the graph in Figure 5, in addition to l_2, l_3, and l_4, it is not necessary to save l_0 either. In general, for a finite graph, there is a minimal number of nodes to save in the passed list in order to guarantee termination. However the trade-off of the space-saving strategy may be increased time-consumption. Consider the same graph of Figure 5. If node

l_0 is not present in the passed list, it will be explored again whenever l_3 is explored. This can be avoided by saving l_0 when it is first visited. But the difference from saving l_1 is that saving l_0 is for efficiency and l_1 for termination.

Now we again recall the abstract reachability algorithm in Figure 1 for timed systems. To ensure termination and also to avoid repeated exploration of states (that have more than one predecessors), it saves every new encountered state (l, D) in the passed list when the inclusion-checking for $D \subseteq D'$ fails (i.e. $D \not\subseteq D'$). Obviously this is not necessary if all the predecessors of (l, D) already exist in the PASSED-list. Similar to the case for finite graphs, for termination, we need to save only one state for every *dynamic loop* of a timed automaton.

Definition 4.1 (Dynamic Loops) *Assume a timed automaton with an initial state (l_0, D_0). The set of symbolic states $L_d = \{(l_1, D_1) \ldots (l_n, D_n)\}$ is a dynamic loop of the timed automaton if $(l_1, D_1) \rightsquigarrow (l_2, D_2) \ldots (l_{n-1}, D_{n-1}) \rightsquigarrow (l_n, D_n)$ and $(l_n, D_n) \rightsquigarrow (l_1, D_1')$ with $D_1' \subseteq D_1$, and (l_0, D_0) is reachable in the sense that $(l_0, D_0) \rightsquigarrow \ldots \rightsquigarrow (l_1, D_1)$. A symbolic state is said to cover a dynamic loop if it is a member of the loop.* □

We claim that to ensure termination, it is sufficient (but not necessary) to save a set of symbolic states that cover all the dynamic loops. Now, the problem is how to compute efficiently such a set.

4.2 Control Structure Analysis and Application

We shall utilise the statical structure of an automaton to identify potential candidates of states to cover dynamic loops.

Definition 4.2 (Statical Loops and Entry Nodes) *A set of nodes $L = \{l_1, \ldots, l_n\}$ of a timed automaton is a statical loop if there is a sequence of edges $l_1 \Leftrightarrow l_2 \cdots l_{n-1} \Leftrightarrow l_n$ and $l_n \Leftrightarrow l_1$ where $l_i \Leftrightarrow l_j$ denotes that $l_i \overset{g,r}{\Leftrightarrow} l_j$ for some g, r is an edge of the automaton. A node $l_i \in L$ is an entry node of the statical loop L if it is an initial node of the automaton or there exists a node $l \notin L$ (outside of the loop) and an edge $l \Leftrightarrow l_i$. Further, we say that a vector of nodes (i.e. a node of a network) is an entry node if any of its components are entry nodes.* □

For example, nodes l_0, l_1, l_2 and l_3 in Figure 5 constitute a statical loop with entry nodes l_0 and l_1; another statical loop is nodes l_1 and l_4 with entry node l_1. In general, since the sets of control nodes and edges of a timed automaton are finite, the number of statical loops is finite and so is the set of entry nodes of all statical loops. In fact the set of entry nodes of a timed automaton can be easily computed by statical analysis using a stack or a slightly modified loop detecting algorithm (see e.g. [Sed88]).

Now note that according to Definition 4.1, a dynamic loop (a set of symbolic states) must contain a subset of symbolic states whose control nodes constitute a statical loop. As a statical loop always contains an entry node, we have the following fact.

Proposition 4.1 *Every dynamic loop of a timed automaton contains at least one symbolic state (l, D) where l is an entry node.*

PROOF: Standard proof by contradiction. □

Following Proposition 4.1, to cover all the dynamic loops, we may simply save all the states whose control-nodes are an entry node, and ignore the others. Obviously, this will not give much reduction when dynamic loops include mostly entry nodes, which is the case when a network of automata contains a component whose nodes are mostly entry nodes e.g. a testing automaton. For networks of automata, we adopt the strategy of saving the *first derived* states whose control nodes are an entry node, known as *covering states* in the following sense.

Definition 4.3 (Covering States) *Assume a network of timed automata with an initial state (l_0, D_0) and a given symbolic state (l, D). We say that (l, D) is a covering state of the network if it is reachable in the sense that there exists a sequence of symbolic transitions $(l_0, D_0) \rightsquigarrow (l_1, D_1) \ldots (l_n, D_n) \rightsquigarrow (l, D)$ and an i (standing for the ith component of the network) such that $l[i]$ is an entry node and $(l_n, D_n) \rightsquigarrow (l, D)$ is derived by an edge $l_n[i] \stackrel{g,r}{\Rightarrow} l[i]$ for some g and r.* □

From the above definition, it should be obvious that we can easily decide whether a reachable symbolic state is a covering state by an on-the-fly algorithm when the entry nodes of all the component automata are known through statical analysis as discussed earlier.

Finally, we claim that the set of *covering* states of a network covers all its dynamic loops and therefore it suffices to keep them in the passed list for the sake of termination in reachability analysis[7].

Theorem 4.1 *Every dynamic loop of a network of timed automata contains at least one covering state.*

PROOF: Assume a dynamic loop $L_d = (l_1, D_1) \rightsquigarrow \ldots \rightsquigarrow (l_k, D_k)$ with no covering states. However according to Proposition 4.1, L_d contains at least one entry node. Further, assume (without loss of generality) that the symbolic state $(l, D) \in L_d$ is an entry node and the components $l[1], \ldots, l[m]$ of l are all in an entry node, and all the other components of l, i.e. $l[m+1], \ldots, l[n]$, are not.

Now, we claim that if L_d contains no covering states, the set of components $l_i[1], \ldots, l_i[m]$ will remain in an entry node in all symbolic states $(l_i, D_i) \in L_d$. Otherwise, if the set of local entry nodes changes, either grows or reduces, it will introduce a covering state. The case of growing is obvious due to the definition for covering states. The argument for the case of reducing is the same as the control nodes of all the components will reach l_1 again by the end of L_d, meaning that the set will sooner or later grows again.

In fact, the assumption that L_d contains no covering states, implies an even stronger property, that is, all symbolic transitions in L_d are derived by components in $l_i[m+1], \ldots, l_i[n]$. A transition is derived by a local transition of a component

[7]Note that this is only a sufficient condition but not necessary.

in $l[1], \ldots, l[m]$, means that the set of local entry nodes will either grow or reduce (discussed above) or the local transition leaves the current entry node and enters an another entry node. The later case implies that the new entry node is a covering state.

Now we construct L'_d by removing $l_i[1], \ldots, l_i[m]$ from all symbolic states $(l_i, D_i) \in L_d$, that is, L'_d contains only the components that are not in an entry nodes. Obviously, all the symbolic transitions of L_d are also in L'_d; thus L'_d must be a loop by definition. However, L'_d contains no components that are in an entry node. This contradicts Proposition 4.1. □

An improved reachability algorithm according to the saving strategy induced from Theorem 4.1 (i.e. saving only the covering sates in the passed list) has been implemented in UPPAAL. Our experimental results show that the space-reduction is between 13–72% (see Table 1 and 2 in Section 5).

5 Experimental Results

The techniques developed in preceding sections have been implemented and added to the tool UPPAAL [BLL+96]. In this section we present the results of an experiment where both the original version of UPPAAL and its extension were applied to verify the following six well-studied examples from the literature:

Philips Audio Protocol (Audio) The protocol was developed and implemented by Philips to exchange control information between components in audio equipment using Manchester encoding. The correctness of the encoding relies on timing delays between signals. It is first studied and manually verified in [BPV94].

We have verified that the main correctness property holds of the protocol, i.e. all bit streams sent by the sender are correctly decoded by the receiver [LPY95b], if the timing error is ±5%.

Philips Audio Protocol with Bus Collision (Audio w. Collision) This is an extended variant of Philips audio control protocol with bus collision detection [BGK+96]. It is significantly larger than the version above since several new components (and variables) are introduced, and existing components are modified to deal with bus collisions.

In the experiment we checked that correct bit sequences are received by the receiver (i.e. Property 1 of [BGK+96]), using the error tolerances set by Philips.

Bang & Olufsen Audio/Video Protocol (Bang & Olufsen) This is an audio control protocol highly dependent on real-time. The protocol is developed by Bang & Olufsen, to transmit messages between audio/video components over a single bus, and further studied in [HSLL97].

In the experiment we have verified the correctness criteria of the protocol. We refer the reader to Section 5.1 of [HSLL97] for more details.

Box Sorter (Box Sorter) The example of [LPY97] is a model of a sorter unit that sorts red and blue boxes. When the boxes moves down a lane they pass a censor

	Current	Local		Global		Local+Global	
	#	#	%	#	%	#	%
Audio	828	219	26	774	93	206	25
Audio w. Collision	646 092	198 178	31	370 800	57	111 632	17
Bang & Olufsen	778 288	249 175	32	642 752	83	204 795	26
Box Sorter	625	139	22	175	28	36	6
Manufact. Plant	92 592	27 042	29	50 904	55	14 933	16
Mutex 2	225	44	20	99	44	18	8
Mutex 3	3 376	621	18	1 360	40	240	7
Mutex 4	56 825	9 352	16	22 125	39	3 532	6
Mutex 5	1 082 916	158 875	15	416 556	38	59 720	6
Train Crossing	464	130	28	384	83	114	25

Table 1: Space performance statistics: number of constraints (#) and percentage of Current (%).

and a piston. The sorter reads the information from the censor and sorts out the red boxes by controlling the position of the piston. We have shown, using UPPAAL, that only blue boxes arrive at the end of the lane.

Manufacturing Plant (Manufact. Plant) The example is a model of the manufacturing plant of [PV94, DY95]. It is a production cell with: a 50 feet belt moving from left to right, two boxes, two robots and a service station. Robot A moves boxes off the rightmost extreme of the belt to the service station. Robot B moves boxes from the service station to the left-most extreme of the belt.

Assuming an initial distance between the boxes on the belt we verified that no box will fall off the belt.

Mutual Exclusion Protocol (Mutex 2–Mutex 5) It is the so-called Fischer's protocol that has been studied previously in many experiments, e.g. [AL92, Sha93]. The protocol is to ensure mutual exclusion among several processes competing for a critical section using timing constraints and a shared variable. In the experiment we use the version of the protocol where a process may recover from failed attempts to enter the critical section, and also eventually leave the critical section [KLL+97].

The protocol is shown to enjoy the invariant property: There is never more than one process existing in the critical section. The results for 2 to 5 processes are shown in Table 1 and 2.

Train Crossing Controller (Train Crossing) It is a variant of the train gate controller [HHWT95]. An approaching train signals to the controller which reacts by closing the gate. When the train have passed the controller opens the crossing. We have verified that the gate is closed whenever a train is close to the crossing.

In Table 1 and 2 we present the space (in number of timing constraints stored on the PASSED-buffer) and in the time requirements (in seconds) of the examples on

	Current	Local		Global		Local+Global	
	sec	sec	%	sec	%	sec	%
Audio	0.44	0.43	98	0.44	100	0.47	107
Audio w. Collision	3 465.22	2 067.37	60	1 515.88	44	929.22	27
Bang & Olufsen	13 240.49	6 967.38	53	9 348.48	71	4 966.79	38
Box Sorter	0.20	0.18	90	0.41	205	0.41	205
Manufact. Plant	155.61	39.85	26	56.61	36	24.22	16
Mutex 2	0.13	0.14	108	0.15	115	0.14	108
Mutex 3	1.40	0.67	48	0.65	46	0.51	36
Mutex 4	102.49	24.48	24	25.97	25	12.14	12
Mutex 5	14 790.56	3 299.96	22	3 111.21	21	1 138.32	8
Train Crossing	0.19	0.18	95	0.20	105	0.18	95

Table 2: Time performance statistics: seconds (sec) and percentage of Current (%).

a Sun SPARCstation4 equipped with 64 MB of primary memory. Each example was verified using the current algorithm of UPPAAL (Current), and using modified algorithms for: Compact Data Structure for Constraints (Local), Control Structure Reduction (Global), and their combination (Local+Global).

As shown in Table 1 and 2 both techniques give truly significant space savings: Compact Data Structure for Constraints saves 68–85% of the original consumed space while Control Structure Reduction demonstrates more variation saving 13–72%. Both methods result in better time-performance on the examples consuming more than half a second, whereas the time-performance is worse on the smaller examples. Most significant is that the two techniques are completely orthogonal, witnessed by the numbers for the combined technique which shows a space-saving between 75% and 94%.

6 Conclusion

In this paper, we have two contributions to the development of efficient data structures and algorithms for memory-usage reduction in the automated analysis of timed systems.

First, we have presented a compact data structure, for representing the subsets of Euclidean space that arise during verification of timed automata, which provides *minimal* and *canonical* representations for clock constraints, and also allows for efficient inclusion checks between constraint systems. The data structure is based on an $\mathcal{O}(n^3)$ algorithm which, given a constraint systems over real-valued variables consisting of bounds on differences, constructs an equivalent system with a minimal number of constraints. It is essentially a minimisation algorithm for weighted directed graphs, that extends the transitive reduction algorithm of [AGU72] to weighted graphs. Given a weighted, directed graph with n vertices, it constructs in time $\mathcal{O}(n^3)$ a reduced graph with the minimal number of edges having the same shortest path closure as the original graph.

Secondly, we have developed an on-the-fly reduction technique to minimise the space-usage by reducing the total number of symbolic states to save in reachability analysis for timed systems. The technique is based on the observation that to ensure termination in reachability analysis, it is not necessary to save all the explored states in memory, but only certain critical states. Based on static analysis of the control structure of timed automata, we are able to compute a set of *covering states* that cover all the dynamic loops of a system. The set of covering states may not be minimal but sufficient to guarantee termination in an on-the-fly reachability algorithm.

The two techniques and their combination have been implemented in the tool UPPAAL. Our experimental results demonstrate that the techniques result in truly significant space-reductions: For a number of well-studied examples in the literature the space saving is between 75% and 94%, and in all large examples time-performance is improved. Noteworthy is also the observation that the two techniques are completely orthogonal.

As future work, we wish to further study the global on-the-fly reduction technique to identify the *minimal* sets of covering states that ensure termination and also avoid repeated explorations in reachability analysis for timed systems.

References

[ABL98] Luca Aceto, Augusto Bergueno, and Kim G. Larsen. Model Checking via Reachability Testing for Timed Automata. In Bernard Steffen, editor, *Proc. of the 4*th *Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, number 1384 in Lecture Notes in Computer Science, pages 263–280. Springer–Verlag, 1998.

[AD90] Rajeev Alur and David Dill. Automata for Modelling Real-Time Systems. In *Proc. of Int. Colloquium on Algorithms, Languages and Programming*, number 443 in Lecture Notes in Computer Science, pages 322–335, July 1990.

[AGU72] A.V. Aho, M.R. Garey, and J.D. Ullman. The Transitive Reduction of a Directed Graph. *SIAM Journal on Computing*, 1(2):131–137, June 1972.

[AL92] Martin Abadi and Leslie Lamport. An Old-Fashioned Recipe for Real Time. In *Proc. of REX Workshop "Real-Time: Theory in Practice"*, number 600 in Lecture Notes in Computer Science, 1992.

[AMP97] Eugene Asarin, Oded Maler, and Amir Pnueli. Data–structures for the Verification of timed automata. In *Proc. of the Int. Workshop on Hybrid and Real-Time Systems*, 1997.

[And95] Henrik Reif Andersen. Partial Model Checking. In *Proc. of Symp. on Logic in Computer Science*, 1995.

[BCM+90] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic Model Checking: 10^{20} states and beyond. In *Proc. of IEEE Symp. on Logic in Computer Science*, 1990.

[Bel57] Richard Bellman. *Dynamic Programming.* Princeton University Press, 1957.

[BGK+96] Johan Bengtsson, W.O. David Griffioen, Kåre J. Kristoffersen, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. Verification of an Audio Pro-

tocol with Bus Collision Using UPPAAL. In Rajeev Alur and Thomas A. Henzinger, editors, *Proc. of the 8*th *Int. Conf. on Computer Aided Verification*, number 1102 in Lecture Notes in Computer Science, pages 244–256. Springer–Verlag, July 1996.

[BLL+96] Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. UPPAAL in 1995. In *Proc. of the 2*nd *Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, number 1055 in Lecture Notes in Computer Science, pages 431–434. Springer–Verlag, March 1996.

[BPV94] D. Bosscher, I. Polak, and F. Vaandrager. Verification of an Audio-Control Protocol. In *Proc. of Formal Techniques in Real-Time and Fault-Tolerant Systems*, number 863 in Lecture Notes in Computer Science, 1994.

[CFJ93] E. M. Clarke, T. Filkorn, and S. Jha. Exploiting Symmetry in Temporal Logic Model Checking. In *Proc. of the 5*th *Int. Conf. on Computer Aided Verification*, number 697 in Lecture Notes in Computer Science, 1993.

[CGL92] E. M. Clarke, O. Grümberg, and D. E. Long. Model Checking and Abstraction. *Principles of Programming Languages*, 1992.

[Dil89] David Dill. Timing Assumptions and Verification of Finite-State Concurrent Systems. In J. Sifakis, editor, *Proc. of Automatic Verification Methods for Finite State Systems*, number 407 in Lecture Notes in Computer Science, pages 197–212. Springer–Verlag, 1989.

[DY95] C. Daws and S. Yovine. Two examples of verification of multirate timed automata with KRONOS. In *Proc. of the 16*th *IEEE Real-Time Systems Symposium*, pages 66–75. IEEE Computer Society Press, December 1995.

[EJ93] E. A. Emerson and C. S. Jutla. Symmetry and Model Checking. In *Proc. of the 5*th *Int. Conf. on Computer Aided Verification*, number 697 in Lecture Notes in Computer Science, 1993.

[GW91] P. Godefroid and P. Wolper. A Partial Approach to Model Checking. In *Proc. of IEEE Symp. on Logic in Computer Science*, pages 406–415, 1991.

[HHWT95] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. A Users Guide to HYTECH. Technical report, Department of Computer Science, Cornell University, 1995.

[HNSY94] Thomas. A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic Model Checking for Real-Time Systems. *Information and Computation*, 111(2):193–244, 1994.

[Hol91] Gerard Holzmann. *The Design and Validation of Computer Protocols.* Prentice Hall, 1991.

[HSLL97] Klaus Havelund, Arne Skou, Kim G. Larsen, and Kristian Lund. Formal Modeling and Analysis of an Audio/Video Protocol: An Industrial Case Study Using UPPAAL. In *Proc. of the 18*th *IEEE Real-Time Systems Symposium.* IEEE Computer Society Press, December 1997.

[KLL+97] Kåre J. Kristoffersen, Francois Laroussinie, Kim G. Larsen, Paul Pettersson, and Wang Yi. A Compositional Proof of a Real-Time Mutual Exclusion Protocol. In *Proc. of the 7*th *Int. Joint Conf. on the Theory and Practice of Software Development*, April 1997.

[LPY95a] Kim G. Larsen, Paul Pettersson, and Wang Yi. Compositional and Symbolic Model-Checking of Real-Time Systems. In *Proc. of the 16*th *IEEE Real-Time Systems Symposium*, pages 76–87. IEEE Computer Society Press, December 1995.

[LPY95b] Kim G. Larsen, Paul Pettersson, and Wang Yi. Diagnostic Model-Checking for Real-Time Systems. In *Proc. of Workshop on Verification and Control of Hybrid Systems III*, number 1066 in Lecture Notes in Computer Science, pages 575–586. Springer–Verlag, October 1995.

[LPY97] Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a Nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1(1–2):134–152, October 1997.

[Pag96] Florence Pagani. Partial Orders and Verification of Real-Time Systems. In Bengt Jonsson and Joachim Parrow, editors, *Proc. of Formal Techniques in Real-Time and Fault-Tolerant Systems*, number 1135 in Lecture Notes in Computer Science, pages 327–346. Springer–Verlag, 1996.

[Pap94] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[PV94] A. Puri and P. Varaiya. Verification of hybrid systems using abstractions. In *Hybrid Systems Workshop*, number 818 in Lecture Notes in Computer Science. Springer–Verlag, October 1994.

[Sed88] Robert Sedgewick. *Algorithms*. Addison-Wesley, 2*nd* edition, 1988.

[Sha93] N. Shankar. Verification of Real-Time Systems Using PVS. In *Proc. of the 5*th *Int. Conf. on Computer Aided Verification*, number 697 in Lecture Notes in Computer Science. Springer–Verlag, 1993.

[Val90] A. Valmari. A Stubborn Attack on State Explosion. *Theoretical Computer Science*, 3, 1990.

[VW86] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. of Symp. on Logic in Computer Science*, pages 322–331, June 1986.

[YL93] Mihalis Yannakakis and David Lee. An efficient algorithm for minimizing real–time transition systems. In *Proc. of the 5*th *Int. Conf. on Computer Aided Verification*, number 697 in Lecture Notes in Computer Science, pages 210–224, 1993.

[YPD94] Wang Yi, Paul Pettersson, and Mats Daniels. Automatic Verification of Real-Time Communicating Systems By Constraint-Solving. In Dieter Hogrefe and Stefan Leue, editors, *Proc. of the 7*th *Int. Conf. on Formal Description Techniques*, pages 223–238. North–Holland, 1994.

Part III
Case Studies

Paper E

Johan Bengtsson, W. O. David Griffioen, Kåre J. Kristoffersen, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. Automated Analsysis of an Audio-Control Protocol Using UPPAAL. In Rajeev Alur and Thomas A. Henzinger, editors, *Proceedings of 9*th *International Conference on Computer Aided Verification*, number 1102 in Lecture Notes in Computer Science, pages 244–256. Springer–Verlag 1996.

Automated Analsysis of an Audio-Control Protocol Using UPPAAL

Johan Bengtsson*, W. O. David Griffioen†, Kåre J. Kristoffersen‡, Kim G. Larsen‡, Fredrik Larsson*, Paul Pettersson*, and Wang Yi*

* Department of Computer Systems, Uppsala University, Sweden.
Email: {johanb,fredrikl,paupet,yi}docs.uu.se.

† CWI, Amsterdam, The Netherlands. Email: griffoe@cwi.nl.

‡ BRICS, Aalborg University, Denmark. Email: {jelling,kgl}cs.auc.dk.

Abstract. In this paper we present a case-study where the tool UPPAAL is extended and applied to verify an Audio-Control Protocol developed by Philips. The size of the protocol studied in this paper is significantly larger than case studies, including various abstract versions verified of the same protocol without bus collision handling, reported previously in the community of real time verification. We have checked that the protocol will function correctly if the timing error of its components is bound to $\pm 5\%$, and incorrectly if the error is $\pm 6\%$. In addition, using UPPAAL's ability of generating diagnostic traces, we have studied an erroneous version of the protocol actually implemented by Philips in their audio products, and constructed a possible execution sequence explaining a known error.

During the case-study, UPPAAL was extended with the notion of *committed locations*. It allows for accurate modelling of atomic behaviours, and more importantly, it is utilised to guide the state-space exploration of the model checker to avoid exploring unnecessary interleavings of independent transitions. Our experimental results demonstrate truly time and space-savings of the modified model checking algorithm. In fact, due to the huge time and memory-requirement, it was impossible to check a simple reachability property of the protocol before the introduction of committed locations, and now it takes only seconds.

1 Introduction

During the last few years a number of tools for automatic verification of hybrid and real-time systems have emerged, e.g. HYTECH [HHWT95], KRONOS [DY95], Polka [HRP94], RT-Cospan [AK95] and UPPAAL [BLL+95]. These tools have by now reached a state, where they are mature enough for industrial applications. We hope to substantiate the claim by reporting on an industry-size case study where the tool UPPAAL is applied.

We analyse an audio control protocol developed by Philips for the physical layer of an interface bus connecting the various devices e.g. CD-players, amplifier etc. in audio equipments. It uses Manchester encoding to transmit bit sequences of arbitrary length between the components, whose timing errors are bound. A simplified version of the protocol is studied by Bosscher et.al. [BPV94]. It is showed that the protocol is incorrect if the timing error of the components is $\pm\frac{1}{17}$ or greater. The proof is carried out without tool support. The first automatic analysis of the protocol is reported in [HWT95] where HYTECH is applied to check an abstract version of the protocol and automatically synthesise the upper bound on the timing error. Similar

versions of the protocol have been analysed by other tools, e.g. UPPAAL [LPY95] and KRONOS [DY95]. However, all the proofs are based on a simplification on the protocol, introduced by Bosscher *et.al.* in 1994, that only one sender is transmitting on the bus so that no bus collisions can occur. In many applications the bus will have more than one sender, and the full version of the protocol by Philips therefore handles bus collisions. The protocol with bus collision handling was manually verified in [Gri94] without tool support. Since 1994, it had been a challenge for the verification tool developers to automate the analysis on the full version of the protocol.

The first automated proof of the protocol with bus collision handling was presented in 1996 in the conference version of this paper [BGK+96]. It was the largest case study, reported in the literature on verification of timed systems, which has been considered as a primary example in the area (see [CW96, LSW97]). The size of the protocol studied is significantly larger than various simplified versions of the same protocol studied previously in the community, e.g. the node-space is 10^3 times larger than the case without bus collision handling and the number of clocks, variables and channels is also increased considerably.

The major problem in applying automatic verification tools to industrial-size systems is the huge time and memory-usage needed to explore the state-space of a network (or product) of timed automata, since the verification tools must keep information not only on the control structure of the automata but also on the clock values specified by clock constraints. It is known as the state–space explosion problem. We experienced the problem right on the first attempt in checking a simple reachability property of the protocol using UPPAAL, which did not terminate in hours though it was installed on a super computer with giga bytes of main memory. We observed that in addition to the size and complexity of the problem itself, one of the main causes to the explosion was the inaccurate modelling of atomic behaviours and inefficient search of the unnecessary interleavings of atomic behaviours by the tool. As a simple solution, during the case-study, UPPAAL was extended with the notion of *committed locations*. It allows for accurate modelling of atomic behaviours, and more importantly, it is utilised to guide the state-space exploration of the model checker to avoid exploring unnecessary interleavings of independent transitions. Our experimental results demonstrate truly time and space-savings of the modified model checking algorithm. In fact, due to the huge time and memory-requirement, it was impossible to check certain properties of the protocol before the introduction of committed locations, and now it takes only seconds.

The automated analysis was originally carried out using an UPPAAL version extended with the notion of committed location installed on a super computer, a SGI ONYX machine [BGK+96]. To make a comparison, we in this paper present an application of the current version of UPPAAL, also supporting committed location, installed on an ordinary Pentium 150 MHz PC machine, to the protocol. We have checked that the protocol will function correctly if the timing error of its components is bound to $\pm 5\%$, and incorrectly if the error is $\pm 6\%$. In addition, using UPPAAL's ability of generating diagnostic traces, we have studied an erroneous version of the protocol actually implemented by Philips in their audio products, and constructed a possible execution sequence explaining a known error.

The paper is organised as follows: In the next two sections we present the UPPAAL model with committed location and describe its implementation in the tool. In section 4 and 5 the Philip Audio-Control Protocol with Bus Collision is informally and formally described. The analysis of the protocol is presented in section 6 where we also compare the performance of the current UPPAAL version with the one used in [BGK+96]. Section 7 concludes the paper. Finally, formal descriptions of the protocol components are enclosed in the appendix.

2 Committed Locations

The basis of the UPPAAL model for real-time systems is networks of timed automata extended with data variables [AD90, HNSY94, YPD94]. However, to meet requirements arising from various case-studies, the UPPAAL model has been extended with various new features such as urgent transitions [BLL+95] etc. The present case-study indicates that we need to further extend the UPPAAL model with *committed locations* to model behaviours such as atomic broadcasting in real-time systems. Our experiences with UPPAAL show that the notion of committed locations introduced in UPPAAL is not only useful in modelling but also yields significant improvements in performance.

We assume that a real-time system consists of a fixed number of sequential processes communicating with each other via channels. We further assume that each communication synchronises two processes as in CCS [Mil89]. Broadcasting communication can be implemented in such systems by repeatedly sending the same message to all the receivers. To ensure atomicity of such "broadcast" sequences we mark the intermediate locations of the sender, which are to be executed immediately, as so-called *committed locations*.

2.1 An Example

To introduce the notion of committed locations in timed automata, consider the scenario shown in Figure 1. A sender S is to broadcast a message m to two receivers R_1 and R_2. As this requires synchronisation between *three* processes this can not directly be expressed in the UPPAAL model, where synchronisation is between two processes with complementary actions. As an initial attempt we may model the broadcast as a sequence of two two-process synchronisations, where first S synchronises with R_1 on m_1 and then with R_2 on m_2. However, this is not an accurate model as the intended atomicity of the broadcast is not preserved (i.e. other processes may interfere during the broadcast sequence). To ensure atomicity, we mark the intermediate location S_2 of the sender S as a *committed location* (indicated by the c:-prefix). The atomicity of the action sequence $\mathsf{m}_1!\mathsf{m}_2!$ is now achieved by insisting that a committed sequence must be left immediately! This behaviour is similar to what has been called "urgent transitions" [HHWT95, DY95, BLL+95], which insists that the next transition taken must be an action (and not a delay), but the essential difference is that no other actions should be performed in between such an atomic sequence. The

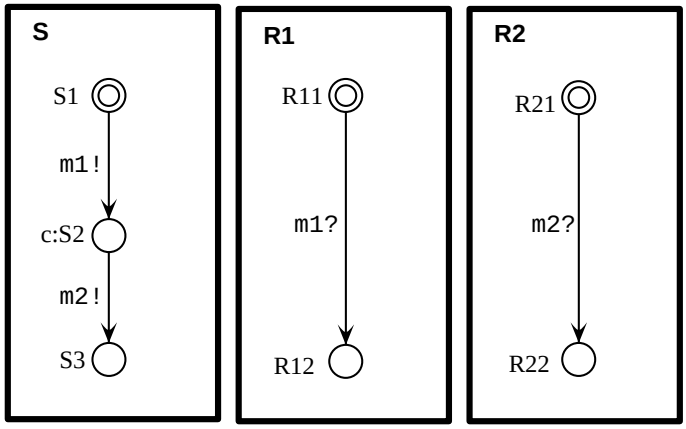


Figure 1: Broadcasting Communication and Committed Locations.

precise semantics of committed locations will be formalised in the transition rules for networks of timed automata with data variables in Section 2.3.

2.2 Syntax

We assume a finite set of clock variables $\mathcal{C}$ ranged over by x, y, z and a finite set of data variables $\mathcal{D}$ ranged over by i, j. We use $\mathcal{B}(\mathcal{C})$ to stand for the set of *clock constraints* that are the conjunctive formulas of simple constraints in the form of $x \prec n$ or $x \Leftrightarrow y \prec n$, where $\prec \in \{<, \leq, =, \geq, >\}$ and n is a natural number. Similarly, we use $\mathcal{B}(\mathcal{D})$ to stand for the set of *non-clock constraints* that are conjunctive formulas of $i \sim j$ or $i \sim k$, where $\sim \in \{<, \leq, =, \neq, \geq, >\}$ and k is an integer number. We use $\mathcal{B}(\mathcal{C},\mathcal{D})$ ranged over by g to denote the set of formulas that are conjunctions of clock constraints and a non-clock constraints. The elements of $\mathcal{B}(\mathcal{C},\mathcal{D})$ are called *constraints* or *guards*.

To manipulate clock and data variables, we use reset-sets which are finite sets of reset-operations. A reset-operation on a clock variable should be in the form $x := n$ where n is a natural number and a reset-operation on an data variable should be in the form: $i := k * j + k'$ where k, k' are integers. A reset-set is a *proper* reset-set when the variables are assigned a value at most once, we use $\mathcal{R}$ to denote the set of all proper reset-sets.

We assume that processes synchronise with each other via complementary actions. Let $\mathcal{A}$ be a set of action names with a subset $\mathcal{U}$ of urgent actions on which processes should synchronise whenever possible. We use $\mathcal{A}ct = \{\ \alpha? \mid \alpha \in \mathcal{A}\ \} \cup \{\ \alpha! \mid \alpha \in \mathcal{A}\ \} \cup \{\ \tau\ \}$ to denote the set of actions that processes can perform to synchronise with each other, where τ is a distinct symbol representing internal actions. We use $\text{name}(a)$ to denote the action name of a, defined by $\text{name}(\alpha?) = \text{name}(\alpha!) = \alpha$.

An automaton A over actions $\mathcal{A}ct$, clock variables $\mathcal{C}$ and data variables $\mathcal{D}$ is a tuple $\langle N, l_0, \Leftrightarrow, I, N_C \rangle$ where N is a finite set of locations (control-locations) with a subset $N_C \subseteq N$ being the set of committed locations, l_0 is the initial location, $\Leftrightarrow \subseteq\ N \times \mathcal{B}(\mathcal{C}, \mathcal{D}) \times \mathcal{A}ct \times \mathcal{R} \times N$ corresponds to the set of edges, and $I : N \mapsto \mathcal{B}(\mathcal{C})$ is the invariant assignment function. To model urgency, we require that the guard

of an edge with an urgent action is a non-clock constraint, i.e. if name$(a) \in \mathcal{U}$ and $\langle l, g, a, r, l' \rangle \in \leftrightarrow\!\!\!\!\rightarrow$ then $g \in \mathcal{B}(\mathcal{D})$.

In the case, $\langle l, g, a, r, l' \rangle \in \leftrightarrow\!\!\!\!\rightarrow$ we shall write $l \overset{g\,a\,r}{\leftrightarrow\!\!\!\!\rightarrow} l'$ which represents a transition from the location l to the location l' with guard g, action a to be performed, and a sequence of reset-operations r to update the variables. Furthermore, we shall write $C(l)$ whenever $l \in N_C$.

To model networks of processes, we introduce a CCS-like parallel composition operator for automata. Assume that $A_1, ..., A_n$ are automata. We use $\overline{A}$ to denote their parallel composition. The intuitive meaning of $\overline{A}$ is similar to the CCS parallel composition of $A_1, ..., A_n$ with *all* actions being restricted, that is, $\overline{A} = (A_1|...|A_n)\backslash\mathcal{A}ct$. Thus only synchronisation between the components A_i is possible. We call $\overline{A}$ a *network of automata*. We simply view $\overline{A}$ as a vector and use A_i to denote its ith component.

2.3 Semantics

Informally, a process modelled by an automaton starts at location l_0 with all its variables initialised to 0. The values of the clocks may increase synchronously with time at location l as long as the invariant condition $I(l)$ is satisfied. At any time, the process can change location by following an edge $l \overset{g\,a\,r}{\leftrightarrow\!\!\!\!\rightarrow} l'$ provided the current values of the variables satisfy the enabling condition g. With this transition, the variables are updated by r.

To formalise the semantics we shall use variable assignments. A *variable assignment* is a mapping which maps clock variables $\mathcal{C}$ to the non-negative reals and data variables $\mathcal{D}$ to integers. For a variable assignment u and a delay d, $u \oplus d$ denotes the variable assignment such that $(u \oplus d)(x) = u(x) + d$ for a clock variable x and $(u \oplus d)(i) = u(i)$ for any data variable i. This definition of $\oplus$ reflects that all clocks proceed at the same speed and that data variables are time-insensitive.

For a reset-set r (a proper set of reset-operations), we use $r[u]$ to denote the variable assignment u' with $u'(w) = \mathit{Value}(e)_u$ whenever $(w := e) \in r$ and $u'(w') = u(w')$ otherwise, where $\mathit{Value}(e)_u$ denotes the value of e in u. Given a constraint $g \in \mathcal{B}(\mathcal{C}, \mathcal{D})$ and a variable assignment u, $g(u)$ is a boolean value describing whether g is satisfied by u or not.

A *control vector* l of a network $\overline{A}$ is a vector of locations where l_i is a location of A_i. We write $l[l'_i/l_i]$ to denote the vector where the ith element l_i of l is replaced by l'_i. Furthermore, we shall write $C(l)$ whenever $C(l_i)$ for some i.

A *state* of a network $\overline{A}$ is a configuration (l, u) where l is a control vector of $\overline{A}$ and u is a variable assignment. The initial state of $\overline{A}$ is (l^0, u^0) where l^0 is the initial control vector whose elements are the initial locations l^0_i of A_i's and u^0 is the initial variable assignment that maps all variables to 0.

The *semantics of a network* of automata $\overline{A}$ is given in terms of a transition system with the set of states being the configurations. The transition relation is defined by the following three rules, which are standard except that each rule has been augmented with conditions handling control-vectors with committed locations:

- $(l,u) \rightsquigarrow (l[l'_i/l_i], r_i[u])$ if $l_i \overset{g_i \tau r_i}{\Leftrightarrow} l'_i$ and $g_i(u)$ for some l_i, g_i, r_i, and for all k if $C(l_k)$ then $k = i$,
- $(l,u) \rightsquigarrow (l[l'_i/l_i, l'_j/l_j], (r_j \cup r_i)[u])$ if $l_i \overset{g_i \alpha! r_i}{\Leftrightarrow} l'_i$, $l_j \overset{g_j \alpha? r_j}{\Leftrightarrow} l'_j$, $g_i(u)$, $g_j(u)$, and $i \neq j$, for some l_i, l_j, g_i, g_j, α, r_i, r_j, and for all k if $C(l_k)$ then $k = i$ or $k = j$,
- $(l,u) \rightsquigarrow (l, u \oplus d)$ if $I(l)(u)$, $I(l)(u \oplus d)$, $\neg C(l)$ and no $l_i \overset{g_i \alpha? r_i}{\Leftrightarrow}$, $l_j \overset{g_j \alpha! r_j}{\Leftrightarrow}$ such that $g_i(u)$, $g_j(u)$, $\alpha \in \mathcal{U}$, $i \neq j$, l_i, l_j, r_i and r_j.

where $I(l) = \bigwedge_i I(l_i)$.

Intuitively, the first rule describes a local internal action transition in a component, and possibly the resetting of variables. An internal transition can occur if the current variable assignment satisfies the transition guard, and the control-locations of all other components in the network are not committed. Thus, internal transitions can not interrupt other components operating in committed locations.

The second rule describes synchronisation transitions that synchronise two components. It is required that the control-locations of all other components are not committed to prevent the transition from interfering with ongoing atomic (i.e. committed) transition sequences in other components.

The third rule describes delay transitions, i.e. when all clocks increase synchronously with time. Delay transitions are permitted only while the location invariants of all components are satisfied. Delays are not permitted if the control-location of a component in the network is committed, or if an urgent transition (i.e. a synchronisation transition with urgent action) is possible. Note that the guards on urgent transitions are non-clock constraints whose truth-values are not affected by delays.

Finally, we note that the three rules give a semantics where components operating in committed location are required to participate in the next transition, which must be an action transition. Furthermore, transition sequences marked as committed are *instantaneous* in the sense that they happen without duration, and *non-interleaved* (or indivisible) as they are never interfered by other components.

3 Committed Locations in UPPAAL

In this section we present a modified version of the model-checking algorithm of UPPAAL for networks of automata with committed locations.

3.1 The Model-Checking Algorithm

The model-checking algorithm performs reachability analysis to check for invariance properties $\forall\Box\beta$, and reachability properties $\exists\Diamond\beta$, with respect to a local property β of the control locations and the values of the clock and data variables. It combines constraint-solving techniques with on-the-fly generation of the state-space in order to avoid explicit construction of the product automaton and the immediately caused memory problems. The algorithm is based on a partitioning of the (otherwise infinite) state-space into finitely many symbolic states of the form (l, D), where

D is a constraint system (i.e. a conjunction of clock constraints and non-clock constraints). It checks if a symbolic state (l^f, D^f) is reachable from the initial symbolic state (l^0, D^0), where D^0 expresses that all clock and data variables are initialised to 0 [YPD94]. Throughout the rest of this paper we shall simply call (l, D) a state instead of symbolic state.

The algorithm essentially performs a forwards search of the state-space. The search is guided and pruned by two buffers: WAITING, holding states waiting to be explored and PASSED holding states already explored. Initially, PASSED is empty and WAITING holds the single state (l^0, D^0). The algorithm then repeats the following steps:

S1. Pick a state (l, D) from the WAITING buffer.

S2. If $l = l^f$ and $D \wedge D^f \neq \emptyset$ return the answer *yes*.

S3. a. If $l = l'$ and $D \subseteq D'$, for some (l', D') in the PASSED buffer, drop (l, D) and go to step S1.

b. Otherwise, save (l, D) in the PASSED buffer.

S4. Find all successor states (l_s, D_s) reachable from (l, D) in one step and store them in the WAITING buffer.

S5. If the WAITING buffer is not empty then go to step S1, otherwise return the answer *no*.

We will not treat the algorithm in detail here, but refer the reader to [YPD94, BL96].

Note that in step S3.b all explored states are stored in the PASSED buffer to ensure termination of the algorithm. In many cases, it will store the whole state-space of the analysed system which grows exponentially both in the number clocks and components [YPD94]. The algorithm is therefore bound to run into space problems for large systems. The key question is how to reduce the growth of the PASSED buffer.

The use of committed location to model atomic behaviours render possible two potential reductions of the PASSED buffer size. First, as atomic sequences in general restrict the amount of interleaving that is allowed in a system [Hol91], the state-space of the system is reduced, and consequently also the number of states stored in the PASSED buffer. Secondly, as a sequence of committed locations semantically is instantaneous and non-interleaved with other components, it suffices to save only the control-location at the beginning of the sequence in the PASSED buffer to ensure termination. Hence, our proposed solution is simply *not* to save states in the PASSED buffer which involve *committed* locations. We modify step S3 of the algorithm in the following way:

S3′. a. If $C(l)$ go directly to step S4.

b. If $l = l'$ and $D \subseteq D'$, for some (l', D') in the PASSED buffer, drop (l, D) and go to step S1.

c. If neither of the above steps are applicable, save (l, D) in the PASSED buffer.

So, for a given state (l, D), if l is committed the algorithm proceeds directly from step S3′.a to step S4, thereby omitting the time-consuming step S3′.b and the space-consuming step S3′.c. Clearly, this will reduce the growth of the PASSED buffer and

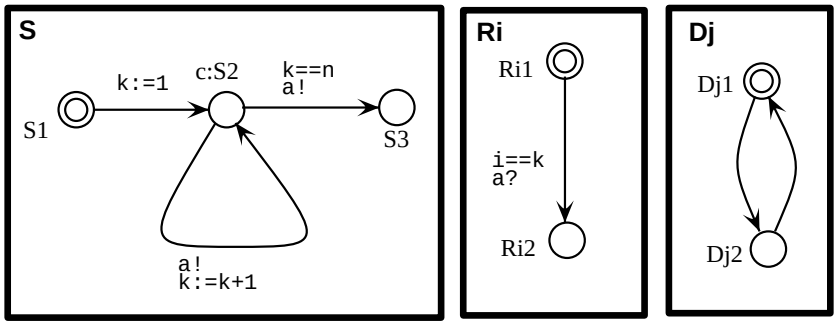


Figure 2: Broadcasting Using Committed Locations.

the total amount of time spent on step S3′. In the following step S4 more reductions are made as interleavings are not allowed when l is committed. In fact, the next transition must be an action transition and it must involve all l_i which are committed in l (according to the transition rules in the previous section). This reduces the time spent on generating successor states of (l, D) in S4 as well as the total number of states in the system. Finally, we note that reducing the PASSED buffer size also yields potential time-savings in step S3′.b when l is *not* committed as it involves a search through the PASSED buffer.

3.2 Space and Time Performance Improvements

To investigate the practical benefits from the usage of committed locations and its implementation in UPPAAL we perform an experiment with a parameterizable scenario, where a sender S wants to broadcast a message to n receivers $\mathsf{R}_1, \ldots, \mathsf{R}_n$. The sender S simply performs n a!-transitions and then terminates, whereas the receivers are all willing to perform a single a?-transition hereby synchronizing with the sender. The data variable k ensures that the ith receiver participates in the ith handshake. Additionally, there are m auxiliary automata $\mathsf{D}_1, \ldots, \mathsf{D}_m$ simply oscillating between two states. Consider Figure 2, where the control node S_2 is committed (indicated by the c:-prefix).

We may now use UPPAAL to verify that the sender succeeds in broadcasting the message, i.e. it forces all the receivers to terminate. More precisely we verify that $\mathsf{SYS}_{n,m} = (\ \mathsf{S}_n \mid \mathsf{R}_1 \mid \ldots \mid \mathsf{R}_n \mid \mathsf{D}_1 \mid \ldots \mid \mathsf{D}_m)$ satisfies the formula $\exists\Diamond(\mathsf{at}(\mathsf{S},\mathsf{S}_3)\ \wedge_{i=1}^{n} \mathsf{at}(\mathsf{R}_i,\mathsf{R}_{i2}))$, where we assume that the proposition $\mathsf{at}(A,l)$ is implicitly assigned to each location l of the automaton A, meaning that the component A is operating in location l. We perform two test sequences, with S_2 declared as respectively not committed and committed. The result is shown in Figure 3. In both test sequences the number of disturbing automata was fixed to eight. Time is measured in seconds and space is measured in pages (4KB). The general observation is that use of committed locations in broadcasting saves time as well as space. The most important observation is that in the committed scenario the space consumption behaves as a constant function in the number of receivers.

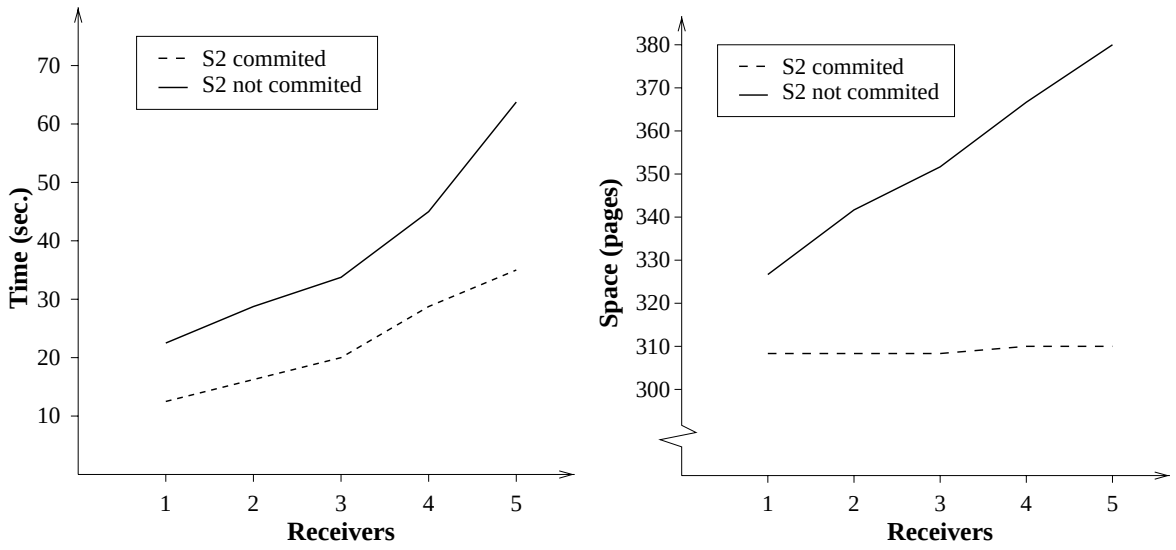


Figure 3: Time and Space Consumption.

4 The Audio Control Protocol with Bus Collision

In this section an informal introduction to the audio protocol with bus collision is given. The audio control protocol is a bus protocol, all messages are received by all components on the bus. If a component receives a message not addressed to it, the message is just ignored. Philips allows up to 10 components.

Messages are transmitted using Manchester encoding. Time is divided into bit-slots of equal length, a bit "1" is transmitted by an up-going edge halfway a bit-slot, a bit "0" by a down-going edge halfway a bit-slot. If the same bit is transmitted twice in a row the voltage changes at the end of the first bit-slot. Note that only a single wire is used to connect the components, no extra clock wire is needed. This is one of the properties that makes it a nice protocol.

The protocol has to cope with some problems: (a) The sender and the receiver must agree on the beginning of the first bit-slot, (b) the length of the message is not known in advance by the receiver, (c) the down-going edges are not detected by the receiver. To resolve these problems the following is required: Messages must start with a bit "1" and messages must end with a down-going edge. This ensures that the voltage on the wire is low between messages. Furthermore the senders must respect a so-called "radio silence" between the end of a message and the beginning of the next one. The radio silence marks the end of a message and the receiver knows that the next up-going edge is the first edge of a new message. It is almost possible to decode a Manchester encoded message by only looking to the up-going messages (problem c) only the last zero bit of a message can not be detected (consider messages "10" and "1"). To resolve this, it is required that all messages are of odd length.

It is possible that two or more components start transmitting at the same time. The behavior of the electric circuit is such that the voltage on the wire will be high as long as one of the senders pulls it high. In other words: The wire implements the or-function. This makes it possible for a sender to notice that someone else is also

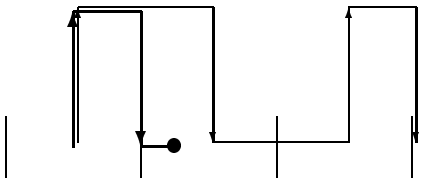


Figure 4: An Example.

transmitting. If the wire is high while it is transmitting a low, a sender can detect a bus collision. This collision detection happens at certain points in time. Just before each up-going transition, and at one and three quarters of a bit-slot after a down going edge (if it is still transmitting a low). When a sender detects a collision it will stop transmitting and will try to retransmit its message later.

If two messages are transmitted at the same time and one is a prefix of the other, the receiver will not notice the prefix message. To ensure collision detection it is not allowed that a message is a prefix of an other message in transit. In the Philips environment this restriction is met by embedding the source address in each message (and assigning each component a unique source address).

In Figure 4 an example is depicted. Assume two senders, named A and B, that start transmitting at exactly the same time. Because two lines on top of each other is hard to distinguish from one line, they are shifted slightly. The sender A (depicted with thick lines) starts transmitting "11..." and sender B (depicted with thin lines) "101...". At the end of the first bit-slot sender A does a down, to prepare for the next up-going edge. But one quarter after this down it detects a collision and stops transmitting. Sender B did not notice the other sender and continues transmitting. Note that the receiver will decode the message of the sender B correctly.

The protocol has to cope with one more thing: timing uncertainty. Because the protocol is implemented on a processor that also has to execute a number of other time critical tasks, a quite large timing uncertainty is allowed. A bit-slot is 888 microseconds, so the ideal time between two edges is 888 or 444 microseconds. On the generation of edges a timing uncertainty of $\pm 5\%$ is allowed. That is, between 844 and 932 for one bit-slot and between 422 and 466 for half a bit-slot. The collision detection just before an up-going edge and the actual generation of this up-going edge must be at most 20 microseconds. The timing uncertainty on the collision detection on one and three quarters after the generation of a down-going edge is ± 22 microseconds. Also the receiver has a timing uncertainty of $\pm 5\%$. And, to complete the timing information, the distance between the end of one message and the beginning of the next must be at least 8000 microseconds (8 milliseconds).

5 A Formal Model of the Protocol

To analyse the behavior of the protocol we model the system as a network of seven timed automata. The network consists of two parts: a *core part* and a *testing environment*. The core part models the components of the protocol to be implemented: two senders, a wire and a receiver. The testing environment, consisting of two message generators and an output checker, is used to model assumptions about the environ-

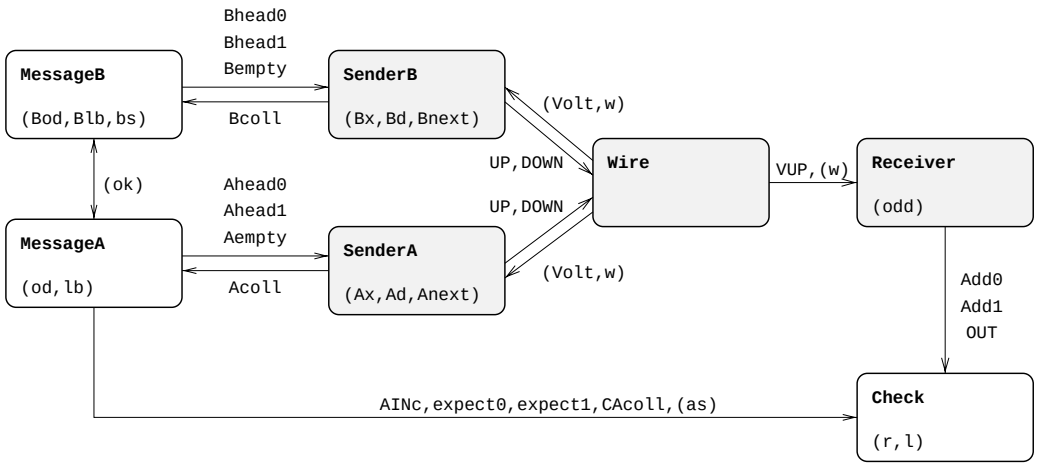


Figure 5: Philips Audio-Control Protocol with Bus Collision.

ment of the protocol and for testing the behavior of the core part. Figure 5 shows a flow-graph of the network where nodes represent timed automata and edges represent synchronisation channels or shared variables (enclosed within parenthesis).

The general idea of the model is as follows. The two automata MessageA and MessageB generate messages for the both senders, in addition MessageA informs the Check-automaton on the bits it generated for SenderA. The senders transmit the messages via the wire to the receiver. The receiver communicates the bits it decoded to the checker. Thus the Check automaton is able to compare the bits generated by MessageA and the bits received by Receiver. If this matches the protocol is correct.

The senders A and B are, modulo renaming (all A's in identifiers to B's), exactly the same. Because of this symmetry, it is enough to check that the messages transmitted by sender A are received correctly. We will proceed with a short description of each automaton. The definition of these uses a number of constants that are declared in Table 1 in Appendix A.

The Senders

SenderA is depicted in Figure 9. It takes input actions Ahead0?, Ahead1? and Aempty?. The output actions UP! and DOWN! will be the Manchester encoding of the message. The clock Ax is used to measure the time between UP! and DOWN! actions. The idea behind the model (taken from [DY95]) is that the sender changes location each half of a bit-slot. The locations HS (wire is High in Second half of the bit-slot) and HF (High in First half of the bit-slot) refer to this idea. Extra locations are needed because of the collision detection.

The clock Ad is used to measure the time elapsed between the detection just before UP! action and the corresponding UP! action. The system is in the locations ar_Qfirst and ar_Qlast when the next thing to do is the collision test at one or three quarters of a bit-slot. When Volt is greater than zero, at that moment, the sender detects a collision, stops transmitting and returns to the idle location. The clock w is used to ensure the radio silence between messages. This variable is checked on the

transition from idle to ar_first_up.

The Wire

This small automaton keeps track of the voltage on the wire and generates VUP! actions when appropriate, that is when a UP? action is received when the voltage is low. The automaton is shown in Figure 10.

The Receiver

Receiver, shown in Figure 8, decodes the bit sequence using the up-going (modeled as VUP?) changes of the wire. Decoded bits are signaled to the environment using output actions Add0!, Add1! and OUT! (where OUT! is used for signaling the end of a decoded message). The decoding algorithm of the receiver is a direct translation of the algorithm in the Philips documentation of the protocol. In the automaton each VUP? transition is followed by a transition modeling the decoding. This decoding happens at once, therefore the intermediate locations are modeled as committed locations. The automaton has two important locations, L1 and L0. When the last received bit is a bit "1" the receiver is in location L1, after receiving a bit "0" it will be in location L0. The error location is entered when a VUP? is received much to early. In the complete model the error location is not reachable, see Section 6. The receiver keeps track of the parity of the received message using the integer variable odd. When the last received bit is a bit "1" and the message is even, a bit "0" is added to make the complete message of odd length.

The Message Generators

The message generators MessageA and MessageB, shown in Figure 11, generate messages of odd length for sender A and B respectively. Furthermore, the messages generated for sender A are communicated to the checker. The start of a message is signaled to the checker by AINc!, bits by expect0! and expect1!. When a collision is detected by sender A this is communicated to MessageA via Acoll?. The message generator will communicate this on his turn to the check automaton via CAcoll!.

Generating messages of odd length is quite simple. The only problem is that it is not allowed that a message for one sender is a prefix of the message for the other sender. To be more precise: If only one sender is transmitting there is no prefix restriction. Only when the two senders start transmitting at the same time, it is not allowed that one sender transmits a prefix of the message transmitted by the other. As mentioned before the reason for this restriction is that the prefix message is not received by the receiver and it is possible that the senders do not notice the collision. In other words: the prefix message can be lost.

The Checker

This automaton is shown in Figure 7. It keeps track of the bits "in transit", i.e. the bits that are generated by the message generators but not yet decoded by the

receiver. Whenever a bit is decoded or the end of the message is detected not conform the generated message the checker enters location error. Furthermore, when sender A detects a collision the checker returns to its initial location.

6 Verification in UPPAAL

In this section we present the results of analysing the protocol formally described in the previous section. We will use $A.l$ to denote the (implicit) proposition $\mathsf{at}(A, l)$ introduced in Section 3.2. Also, note that invariance properties in UPPAAL are on the form $\forall\Box\beta$, where β is a local property.

Correctness Criteria

The main correctness criterion of the protocol is to ensure that the bit sequence received by the Receiver matches the bit sequence sent by SenderA. Moreover, the *entire* bit sequence should be received by Receiver (and communicated to Check). From the description of the Check-automaton (see the previous section) it follows that this behaviour is ensured if Check is always operating in location start or normal:

$$\forall\Box\ (\mathsf{Check.start} \vee \mathsf{Check.normal}) \tag{1}$$

When the Receiver-automaton observes changes of the wire too early it changes control to location error. If the rest of the components behave normally this should not happen. Therefore, the Receiver-automaton is required to never reach the location error:

$$\forall\Box(\neg\mathsf{Receiver.error}) \tag{2}$$

Incorrectness

Unfortunately the protocol described in this paper is not the protocol that Philips has implemented. The original sender checked less often for a bus collision. The "just before the up going edge" collision detection was only performed before the first up. In the UPPAAL model this comes down to deleting outgoing transitions of ar_Qlast_ok and using the outgoing transitions of ar_up_ok instead. This incorrect version is shown in Figure 12. In general the problem is that if both senders are transmitting and one is slow and the other fast, the distance can cumulate to a high value that can confuse the receiver. UPPAAL generated a counterexample trace to Property 1. The trace is depicted in Figure 6. The scenario is as follows: Sender A (depicted with thick lines) tries to transmit "111..." and sender B (depicted with thin lines) "1100...". The sender A is fast and the other slow. This makes that the distance between the second UP's is quite big (77 microseconds). In the third bit-slot the sender A detects the collision. The result of all this is that the time elapsed between the VUP actions is 6.65Q instead of the ideal 6Q. And because of the timing uncertainty in the receiver this can be interpreted as 7Q ($7 * 0.95 = 6.65$). And 7Q is just enough to decode "01" instead of the transmitted "0". In the correct version this scenario is impossible,

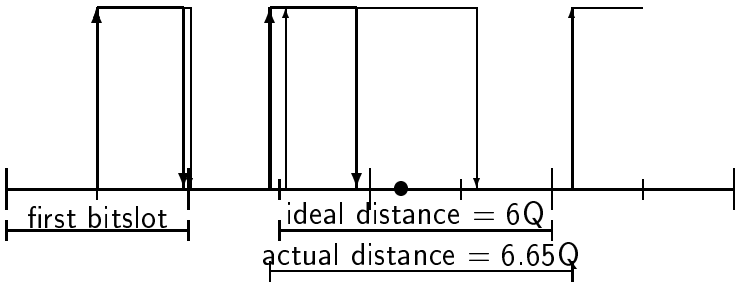


Figure 6: Error execution of the incorrect protocol.

because if collision detection happens before *every* UP action, the distance between the UP's in the second bit-slot can not be that high (at most 20 microseconds).

It is not likely that these kind of errors happen in the actual implementation. This is prevented by, among others, the following: It is not likely that two senders do start at (roughly) the same time. The timing uncertainty is at most 2% instead of 5%. And the "average" timing uncertainty is even less. And finally, the source address is in the beginning of the messages, this makes the senders detect the collision. See [Gri94] for more details.

Although this problem was know by Philips it is interesting to see how powerful the diagnostic traces can be. It enables us not only to find mistakes in the *model* of a protocol, but also to find design mistakes in real life protocols.

Verification Results

UPPAAL successfully verifies the correctness properties 1 and 2 for an error tolerance of 5% on the timing. Recall that SenderA and SenderB are, modulo renaming, exactly the same, implying that the verified properties for SenderA also applies to the symmetric case for SenderB. The verification of Property 1 and 2 was performed in 12.75 sec using 2.1 MB of memory.

The analysis of the incorrect version of the protocol with less collision detection (discussed above) uses UPPAAL's ability to generate diagnostic traces whenever an invariant property is not satisfied by the system. The trace, consisting of 46 transitions, was generated in 4.5 sec using 1.8 MB of memory. Also, attempts to verify Property 1 for the full protocol with an error tolerance of 6% on the timing failed. The scenario is similar to the one found by Bosscher et.al. in [BPV94] for the one sender protocol.

The properties were verified using UPPAAL version 2.17 [LPY97a, BLL+98] that implements the verification algorithm for handling committed locations described in Section 3. It was installed on a Pentium 150 MHz MMX running Red Hat Linux 5.0. In the conference version of this paper [BGK+96] we reported that the same protocol was verified using UPPAAL version 0.96[1] installed on a SGI ONYX machine. The verification of the two correctness properties then consumed 7.5 hrs using 527.4 MB and 1.32 hrs using 227.9 MB, whereas a diagnostic trace for the incorrect version was generated in 13.0 min using 290.4 MB of memory. Hence, both the time- and space-

[1]The two UPPAAL versions 0.96 and 2.17 are dated Nov 1995 and March 1997 respectively.

consumption of the verifier have been reduced with over 99%. These improvements of the UPPAAL verifier are due to a number of developments in the last two years that will not be discussed further here, but we refer the reader to [LPY97b, BLL+98].

7 Conclusions

In this paper we have presented a case-study where the verification tool UPPAAL is applied to analyse a realistic audio-control protocol by Philips with bus collision handling. The protocol has received a lot of attention in the formal methods research community (see e.g. [LSW97, CW96]) and simplified versions of the protocol without the handling of bus collisions have previously been analysed by several research teams, with and without support from automatic tools. To our knowledge, the full protocol considered in this paper has never before been automatically analysed.

As verification results we have shown that the protocol behaves correctly if the error on all timing is bound to $\pm 5\%$, and incorrectly if the error is $\pm 6\%$. Furthermore, using UPPAAL's ability to generate diagnostic traces we have been able to study error scenarios in an incorrect version of the protocol actually implemented by Philips.

In this paper we have also introduced the notion of so-called committed locations which allows for accurate modelling of atomic behaviours. More importantly, it is also utilised to guide the state-space exploration of the model checker to avoid exploring unnecessary interleavings of independent transitions. Our experimental results demonstrate truly time and space-savings of the modified model checking algorithm. In fact, due to the huge time and memory-requirement, it was impossible to check certain properties of the protocol before the introduction of committed locations, and now it takes only seconds.

References

[AD90] Rajeev Alur and David Dill. Automata for Modelling Real-Time Systems. In *Proc. of Int. Colloquium on Algorithms, Languages and Programming*, number 443 in Lecture Notes in Computer Science, pages 322–335, July 1990.

[AK95] Rajeev Alur and Robert P. Kurshan. Timing Analysis in COSPAN. In Rajeev Alur, Thomas A. Henzinger, and Eduardo D. Sontag, editors, *Proc. of Workshop on Verification and Control of Hybrid Systems III*, number 1066 in Lecture Notes in Computer Science, pages 220–231. Springer–Verlag, October 1995.

[BGK+96] Johan Bengtsson, W.O. David Griffioen, Kåre J. Kristoffersen, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. Verification of an Audio Protocol with Bus Collision Using UPPAAL. In Rajeev Alur and Thomas A. Henzinger, editors, *Proc. of the 8*th *Int. Conf. on Computer Aided Verification*, number 1102 in Lecture Notes in Computer Science, pages 244–256. Springer–Verlag, July 1996.

[BL96] Johan Bengtsson and Fredrik Larsson. UPPAAL a Tool for Automatic Verification of Real-time Systems. Master's thesis, Uppsala University, 1996. Available as `http://www.docs.uu.se/docs/rtmv/bl-report.pdf`.

[BLL+95] Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. UPPAAL — a Tool Suite for Automatic Verification of Real–Time Systems. In *Proc. of Workshop on Verification and Control of Hybrid Systems III*, number 1066 in Lecture Notes in Computer Science, pages 232–243. Springer–Verlag, October 1995.

[BLL+98] Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, Wang Yi, and Carsten Weise. New Generation of UPPAAL. In *Int. Workshop on Software Tools for Technology Transfer*, June 1998.

[BPV94] D. Bosscher, I. Polak, and F. Vaandrager. Verification of an Audio-Control Protocol. In *Proc. of Formal Techniques in Real-Time and Fault-Tolerant Systems*, number 863 in Lecture Notes in Computer Science, 1994.

[CW96] Edmund M. Clarke and Jeanette M. Wing. Formal Methods: State of the Art and Future Directions. *ACM Computing Surveys*, 28(4):626–643, December 1996.

[DY95] C. Daws and S. Yovine. Two examples of verification of multirate timed automata with KRONOS. In *Proc. of the 16*th *IEEE Real-Time Systems Symposium*, pages 66–75. IEEE Computer Society Press, December 1995.

[Gri94] W.O. David Griffioen. Analysis of an Audio Control Protocol with Bus Collision. Master's thesis, University of Amsterdam, Programming Research Group, 1994.

[HHWT95] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. HYTECH: The Next Generation. In *Proc. of the 16*th *IEEE Real-Time Systems Symposium*, pages 56–65. IEEE Computer Society Press, December 1995.

[HNSY94] Thomas. A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic Model Checking for Real-Time Systems. *Information and Computation*, 111(2):193–244, 1994.

[Hol91] Gerard Holzmann. *The Design and Validation of Computer Protocols*. Prentice Hall, 1991.

[HRP94] N. Halbwachs, P. Raymond, and Y.-E. Proy. Verification of linear hybrid systems by means of convex approximations. In *Static Analysis Symposium*, number 864 in Lecture Notes in Computer Science, pages 223–237, 1994.

[HWT95] Pei-Hsin Ho and Howard Wong-Toi. Automated Analysis of an Audio Control Protocol. In *Proc. of the 7*th *Int. Conf. on Computer Aided Verification*, number 939 in Lecture Notes in Computer Science. Springer–Verlag, 1995.

[LPY95] Kim G. Larsen, Paul Pettersson, and Wang Yi. Diagnostic Model-Checking for Real-Time Systems. In *Proc. of Workshop on Verification and Control of Hybrid Systems III*, number 1066 in Lecture Notes in Computer Science, pages 575–586. Springer–Verlag, October 1995.

[LPY97a] Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a Nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1(1–2):134–152, October 1997.

[LPY97b] Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL: Status and Developments. In Orna Grumberg, editor, *Proc. of the 9*th *Int. Conf. on Computer Aided Verification*, number 1254 in Lecture Notes in Computer Science, pages 456–459. Springer–Verlag, June 1997.

[LSW97] Kim G. Larsen, Bernard Steffen, and Carsten Weise. Continuous modeling of real-time and hybrid systems: from concepts to tools. *Int. Journal on Software Tools for Technology Transfer*, 1(1–2):64–85, December 1997.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice Hall, Englewood Cliffs, 1989.

[YPD94] Wang Yi, Paul Pettersson, and Mats Daniels. Automatic Verification of Real-Time Communicating Systems By Constraint-Solving. In Dieter Hogrefe and Stefan Leue, editors, *Proc. of the 7*th *Int. Conf. on Formal Description Techniques*, pages 223–238. North–Holland, 1994.

A The System Description

The constants used in the formulas		
q	2220	One quarter of a bit-slot: 222 micro sec
d	200	Detection 'just before' the UP: 20 micro sec
g	220	'Around' 25% and 75% of the bit-slot: 22 micro sec
w	80000	The radio silence: 8 milli sec
t	0.05	The timing uncertainty: 5%
The constants in the automata		
W	w	80000
D	d	200
A1min	q-g	2000
A1max	q+g	2440
A2min	3*q-g	6440
A2max	3*q+g	6880
Q2	2*q	4440
Q2minD	2*q*(1-t)-d	4018
Q2min	2*q*(1-t)	4218
Q2max	2*q*(1+t)	4662
Q3min	3*q*(1-t)	6327
Q3max	3*q*(1+t)	6993
Q5min	5*q*(1-t)	10545
Q5max	5*q*(1+t)	11655
Q7min	7*q*(1-t)	14763
Q7max	7*q*(1+t)	16317
Q9min	9*q*(1-t)	18981
Q9max	9*q*(1+t)	20979

Table 1: Declaration of Constants.

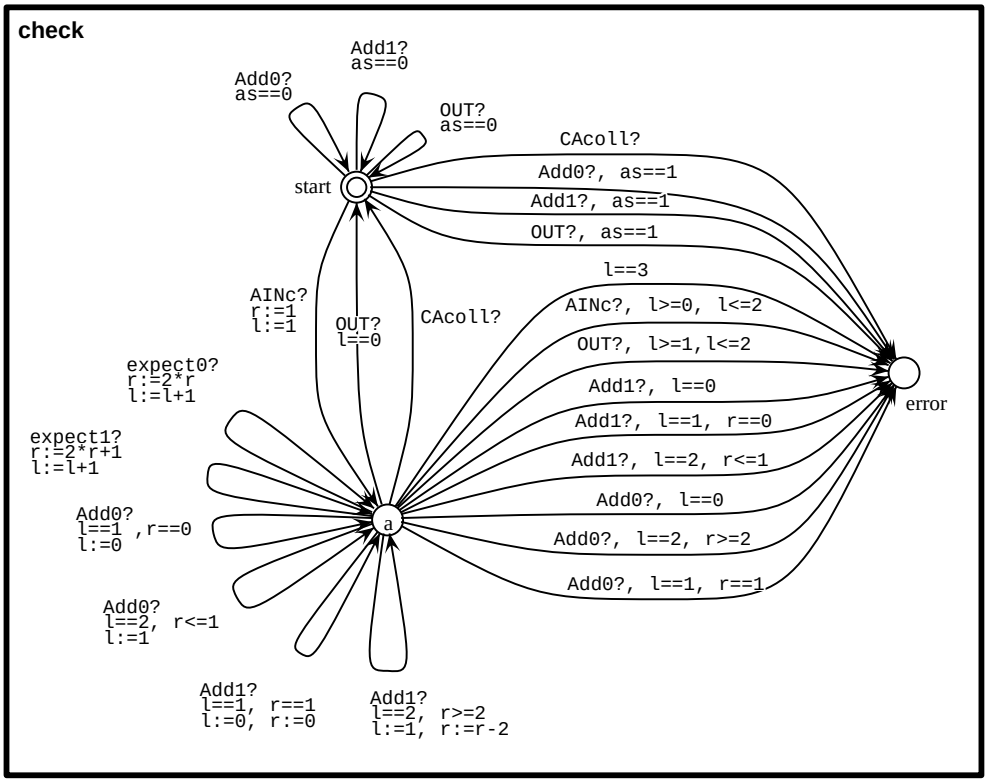


Figure 7: The Check Automaton.

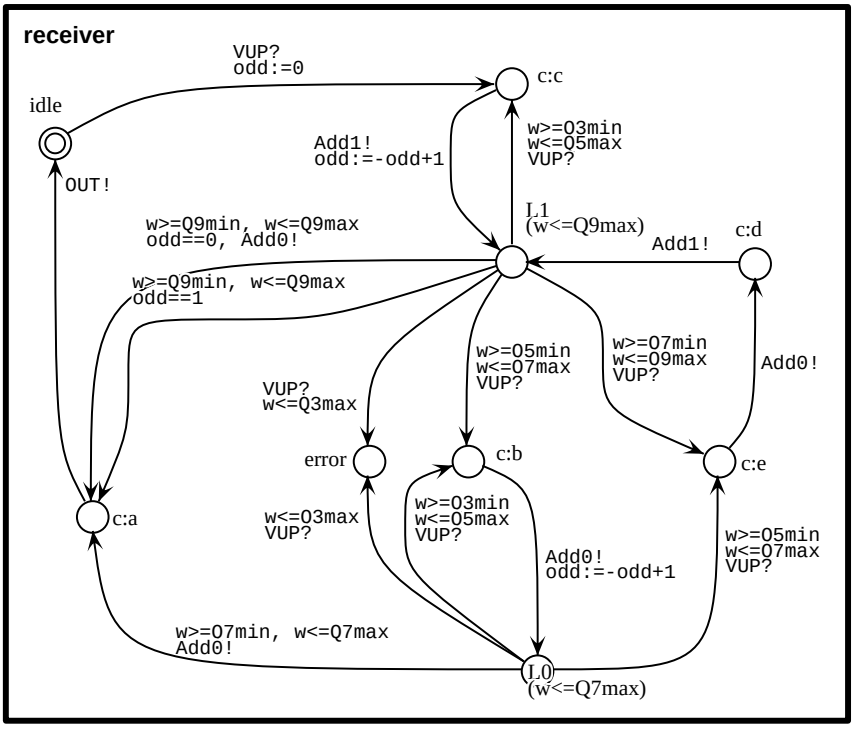


Figure 8: The Receiver Automaton.

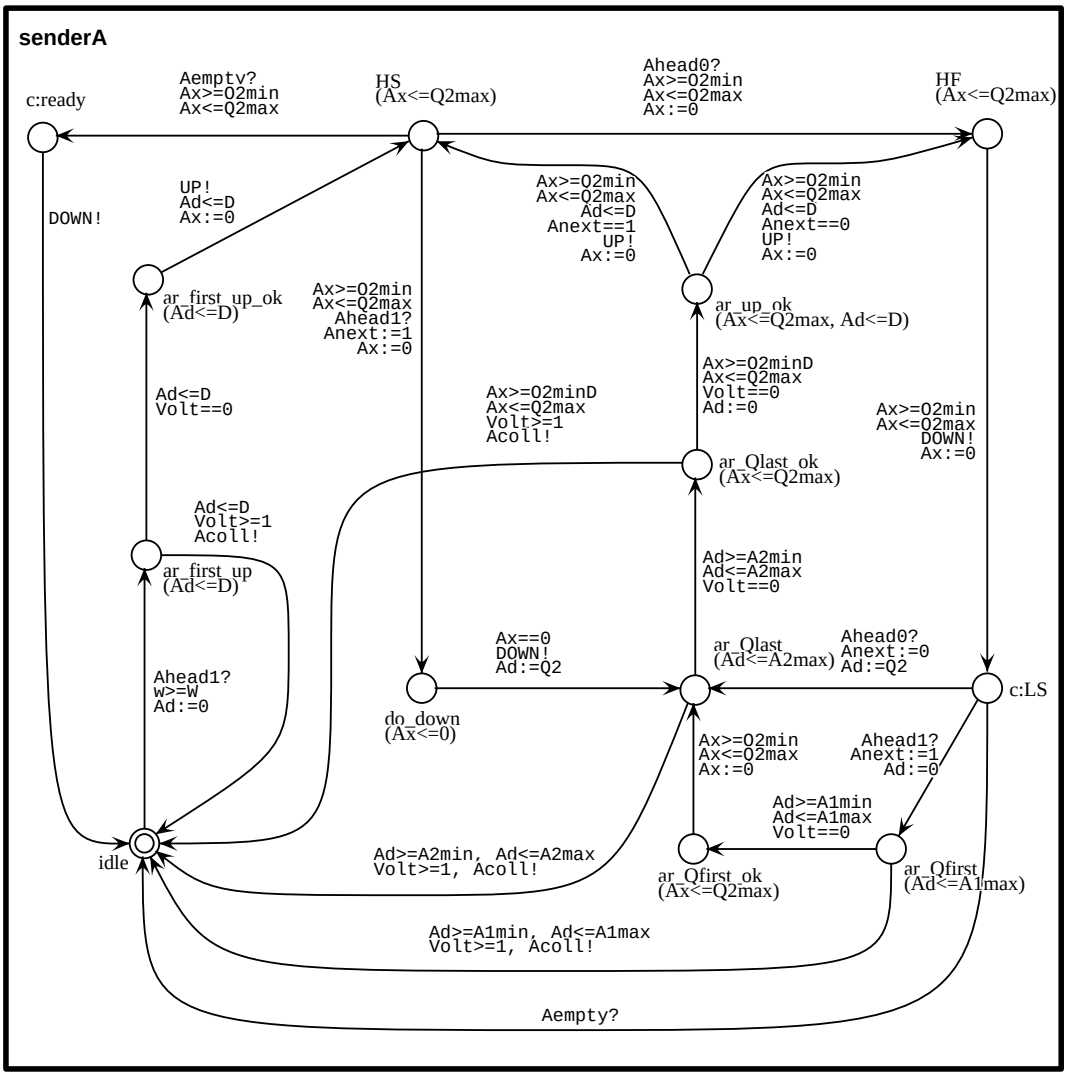


Figure 9: The SenderA Automaton.

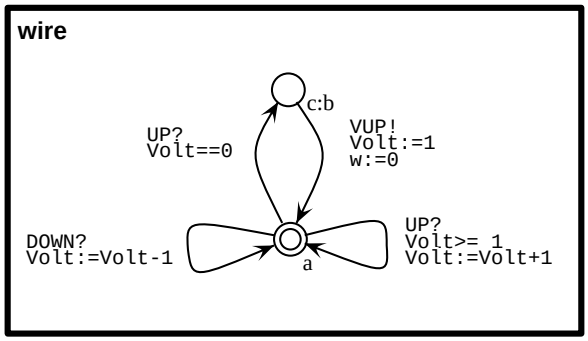
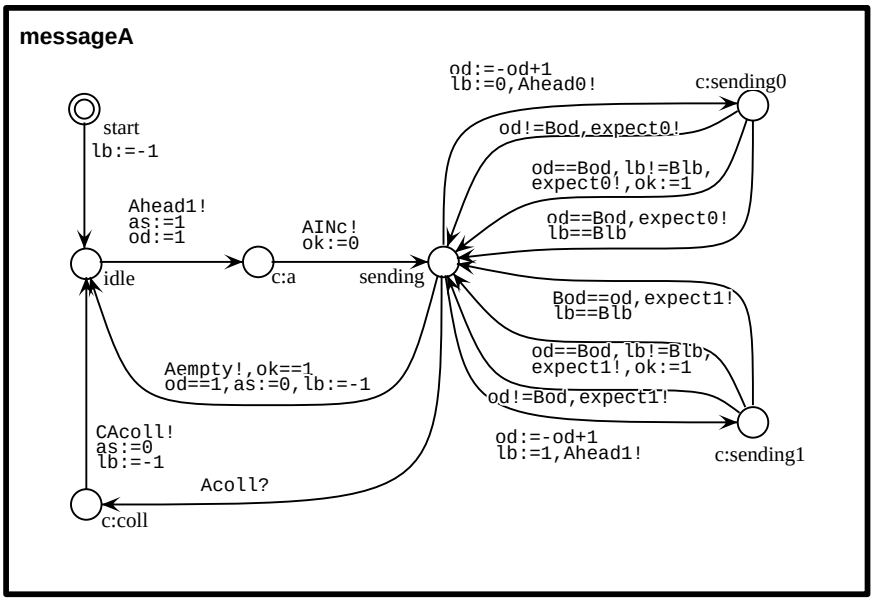


Figure 10: The Wire Automaton.



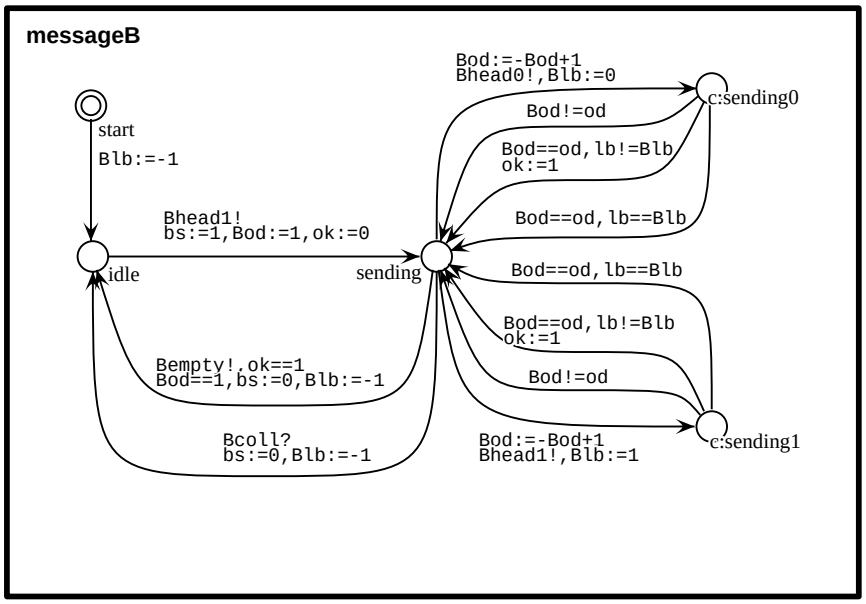


Figure 11: The Message Automata.

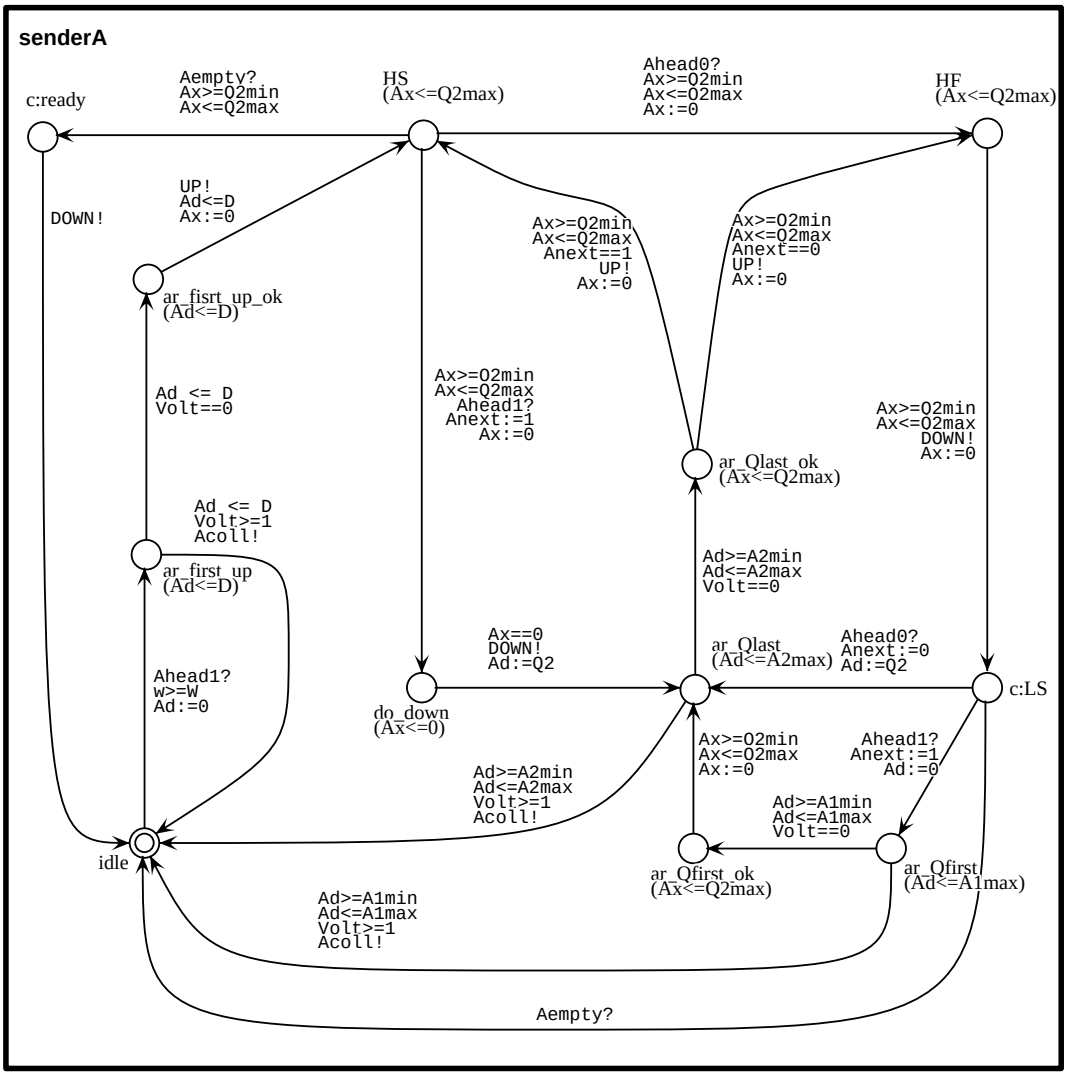


Figure 12: The Incorrect SenderA Automaton.

Paper F

Henrik Lönn and Paul Pettersson. Formal Verification of a TDMA Protocol Startup Mechanism. In *Proceedings of the Pacific Rim International Symposium on Fault-Tolerant Systems*, pages 235–242. IEEE Computer Society Press 1997.

Formal Verification of a TDMA Protocol Startup Mechanism

Henrik Lönn* and Paul Pettersson†

* Department of Computer Engineering, Chalmers University of Technology, Sweden.
Email: hlonn@ce.chalmers.se.

† Department of Computer Systems, Uppsala University, Sweden.
Email: paupet@docs.uu.se.

Abstract.

In this paper we presents a formal analysis of the start-up algorithm of the DACAPO protocol. The protocol uses TDMA (Time Division Multiple Access) bus arbitration. It is checked that a system of four communicating stations becomes synchronised and operational within a bounded time from an arbitrary initial state. The system model allows a clock drift corresponding to $\pm 10^{-3}$. The protocol is modeled as a network of timed automata, and analysis is performed using the symbolic model-checking tool UPPAAL.

1 Introduction

Distributed real-time systems are increasingly used in embedded applications, many of them safety-critical systems, such as cars, aircraft and industrial robots. The computer architecture of such systems must meet several stringent requirements in terms of cost, reliability, testability, etc., and has therefore recently been the object of much research.

One crucial component of the distributed architecture is the communication system. This is the backbone of the system and has a large effect on its quality both in terms of performance and reliability.

Assuming a broadcast bus as the communication media, TDMA-based protocols have several advantages [Kop93]. They are particularly suitable for safety-critical architectures because they facilitate clock synchronisation without any message overhead [BD87] and support timely fault detection. In TDMA protocols, each node has a time slot where it has exclusive access to the bus. Collisions are thus avoided without the frame overhead incurred when using contention-based protocols or the token recovery algorithms needed in token-based protocols.

A characteristic of TDMA protocols is that clocks must be synchronised to guarantee collision-free broadcasts — two nodes may otherwise use the same time slot for transmission. On the other hand, messages must be broadcast to do clock synchronisation.

Since we consider a multi-master system in which all nodes are equals, it is not possible to allow nodes to wait for initialisation messages from a single master. We also believe that the use of a unique "jam" signal for synchronisation [KU95] would make it difficult to detect and isolate a node that erroneously attempts to perform resynchronisation.

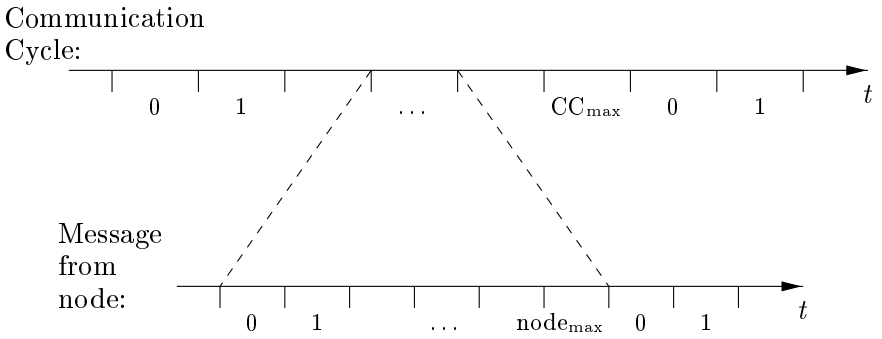


Figure 1: The Cyclic Operation of DACAPO.

The start-up algorithms of the TTP [KG94] and DACAPO [LS94] TDMA protocols have informal descriptions in their respective references. The latter paper also presents simulations that provide estimation on the worst case duration of the start-up phase.

In several applications, the reliability of the real-time system must be better than 10^{-8} failures per hour [Tor92]. This failure probability is so low that validating it by means of simulation is exceedingly time-consuming and yet provides only a probabilistic measure of correctness. An alternative is to formally verify the algorithms.

This paper presents a rigorous description of a TDMA start-up algorithm in the model of timed automata. The algorithm is then analysed for a system consisting of four computers, modelled as a network of timed automata representing the computer ensemble and the bus. It is checked that all stations becomes synchronised within a certain deadline. The UPPAAL [LPY97] tool is used for this purpose.

The paper starts with a brief presentation of the DACAPO protocol and an informal description of how synchronisation is performed. In section 3 we presents a formal model of the protocol. Section 4 gives details on the analysis of the protocol, and conclusions are presented in Section 5.

2 Protocol Description

This section briefly describes the protocol and the real-time architecture for which it is designed. In particular, the start-up algorithm is described.

2.1 General

Dependable Architecture for Control of Applications with Periodic Operation (DACAPO) [RLST95], is a conceptual computer architecture for safety-critical distributed real-time systems. The concept covers the complete computer architecture, but we will focus here on the communication protocol.

The DACAPO protocol is intended for physically small distributed systems. The bus length is limited to tens of meters to avoid problems with large propagation delays, and the number of stations is less than about 40 for reliability reasons. Although

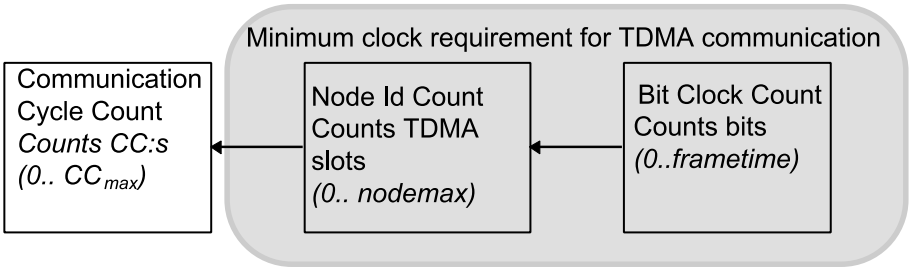


Figure 2: Time Representation in each Station

the DACAPO concept prescribes a duplex bus to meet reliability requirements, this paper considers only operation on a single bus.

The DACAPO protocol is controlled by time only. Time is divided into equally sized time slots, called *TDMA slots*, that are long enough for one message. An operational node broadcasts one message in its own TDMA slot. To minimise hardware complexity and allow simple error detection, time slot i belongs to node i, i.e. the first time slot belongs to node 0, the next to node 1, etc. up to the highest node *id*. When all nodes have broadcast a message, a so-called *communication cycle* is completed. The communication cycle is repeated continuously during system operation, see Figure 1. The node sending order is the same in each communication cycle, but the data contents of a node's message usually change.

Each node thus needs a clock that indicates the duration of each TDMA slot and controls the transmission and reception of bits. We call this the *Bit Clock*, since it is synchronous with the bit stream broadcast on the bus. The Bit Clock will require relatively frequent adjustment owing to clock drift between nodes.

Bits are non return to zero encoded (NRZ), i.e. each bit is transmitted as either a high or low signal. NRZ encoding reduces both bandwidth requirements and noise emission as compared with encoding techniques with guaranteed transitions (such as Manchester encoding). There is no continuous clock adjustment during the reception of a message. Since the messages exchanged in a control application are short — a requirement to limit control delays and allow a sufficiently high sampling frequency of the control loops — adjustments during frame reception can be avoided.

A *Node Id Counter* in each node is used to keep track of the owner of each TDMA slot. The Node Id Counter is incremented each time the bit clock completes one TDMA slot interval.

A minimum requirement of the clock in a TDMA system is shown in Figure 2. The non-shaded *communication cycle count* is not used for the initialisation of communication and is not considered further.

2.2 Bit Synchronisation

With TDMA communication, the arrival time of a message can be used as a clock reading. Since communication is pre-scheduled, the local time of the sending node is implicitly known [BD87]. DACAPO uses the Daisy-Chain synchronisation method [LS95]. With Daisy-Chain synchronisation, the local clock is adjusted on each message ar-

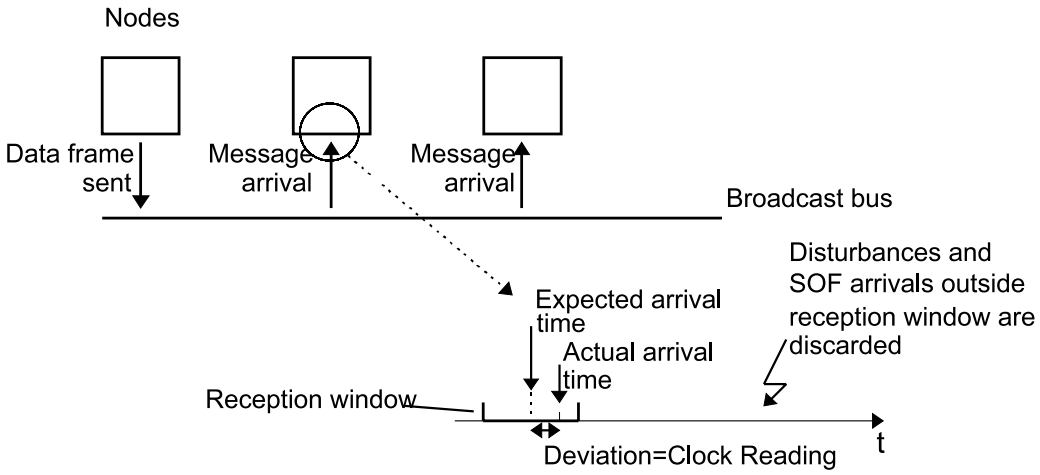


Figure 3: Exchange of Clock Readings.

rival. Since nodes broadcast consecutively according to the TDMA schedule, they take turns in synchronising the system in a Daisy-Chain manner. To avoid partitioning of the system into unsynchronised cliques, only nodes that are synchronised with at least half of the ensemble may transmit messages. Correct synchronisation is decided on the basis of successful reception of a message.

To achieve fault tolerance, a *reception window* is used. Only messages whose Start Of Frame field (SOF), arrives within a narrow interval are accepted (see Figure 3). The size of the reception window equals the length of the SOF field.

2.3 TDMA Time Slot Synchronisation

Synchronising the TDMA time slots is equivalent to synchronising the Node Id Count part of the local clocks. To do this, messages with the sending node's Node Id Count must be broadcast, which is a problem if bus synchronisation has not yet been established. A carefully designed procedure for initial synchronisation of the TDMA time slots is thus necessary.

We use a state vector to describe the components of the internal state of the node that are relevant for the bus synchronisation:

$$\langle \text{ Bitclock count, Node Id Count, Error Count, Mode } \rangle$$

The first two items correspond to the local opinion on system time. The *Error Count* is increased each time an erroneous frame or an empty time slot is detected. It is decreased every time a message is received correctly. Mode is one of three protocol modes local to each node: *Normal, Resynchronisation* or *Recover*.

Node operation in the different internal *Modes* is as follows:

1. **Normal Mode:**
   a. Broadcast frames according to the TDMA schedule.
   b. Assign the Node Id Count of each correctly received frame to the Node Id Count of the node.

   c. Increment Node Id Count by one in case of empty TDMA slot or erroneous frame reception.
   d. Enter Resynchronisation mode if messages from less than half of the nodes ($\lfloor n/2 \rfloor$ out of n) have been received correctly.

2. **Resynchronisation Mode:**
   a. Be Silent.
   b. Assign the Node Id Count of each correctly received frame to the Node Id Count of the node.
   c. Increment Node Id Count by one in case of empty TDMA slot or erroneous frame reception.
   d. Enter Normal Mode as soon as messages from half of the nodes ($\geq \lfloor n/2 \rfloor$ out of n) have been received correctly.
   e. When bus has been completely silent for one Communication Cycle: Enter Recovery Mode.

3. **Recovery Mode:**
   a. Wait for the node's own TDMA slot and broadcast one frame if the node is part of the *recovery set*. If the broadcast occurs in the first slot in Recovery Mode, it is postponed until the next TDMA slot belonging to the node.
   b. Assign the Node Id Count of each correctly received frame to the Node Id Count of the node.
   c. Increment Node Id Count by one in case of empty TDMA slot or erroneous frame reception.
   d. Enter Normal Mode when messages from half of the nodes ($\geq \lfloor n/2 \rfloor$ out of n) have been received correctly.
   e. When one frame has been sent: Enter Resynchronisation Mode. If a collision is detected: reset the local Node Id Count.

In all modes, Node Id Count is set to the id of the sending node on successful reception of a message. This is reasonable, since it is possible to transmit the message without collisions, and the sending node is thus likely to have the correct time.

The condition in 3a, that a node may not broadcast in the first TDMA slot of recovery mode prevents a situation in which a fast node breaks bus silence before all nodes have seen a full Communication Cycle without bus activity. This would prevent the transition from Resynchronisation Mode to Recovery Mode in these nodes. Only the first node would broadcast a message, and no node would receive enough messages to enter normal mode.

The broadcast in 3a may result in an infinite series of collisions for certain system sizes if all nodes broadcast in recover mode. To avoid this, only members of the *recovery set*, a subset of all nodes, may send a message. The nodes that will be members of the recovery set are selected before run-time.

In 3e, the Node Id Count is reset if a collision occurred to avoid a collision between the same stations if the start-up fails and a second transmission in recovery mode is necessary. Together with the recovery set limitation, this makes possible the completion of the start-up procedure without repeated collisions.

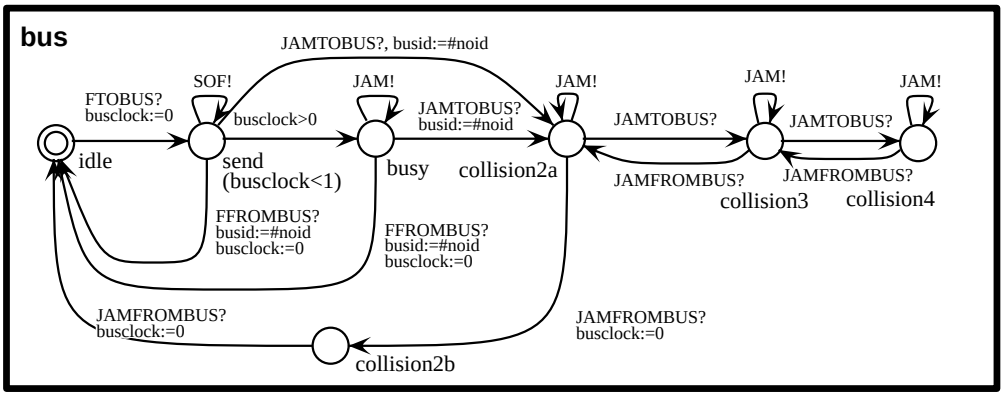


Figure 4: Timed Automaton Describing the Bus.

3 Formal Description of the Protocol

Our formal description of the protocol is defined in the model of timed automata [AD90]. As the UPPAAL tool will be used to analyse the protocol, we use the model of networks of timed automata extended with data variables, adopted in the tool. We will not describe the UPPAAL-model here, but refer the reader to [LPY97].

3.1 Assumptions

Our model is based on the following assumptions regarding the bus communication:

- Clock precision among the station clocks is $\pm 10^{-3}$. The DACAPO protocol is intended for systems with low precision oscillators, and the model must therefore contain a certain clock drift.
- A message is always received correctly if it arrives within the reception window and no collision occurs. We have assumed a clock drift that is sufficiently low to prevent a sending and receiving node from drifting apart during message reception.
- There is no propagation delay on the bus. In a TDMA system that is scheduled before run-time, the propagation delay for each message can be calculated before runtime. Since the sending node is known, compensation can be made for the propagation delay before the local clock is adjusted.
- Broadcast messages are never corrupted unless two or more messages collide. We do not assume any transient or permanent communication faults. Disturbances on the bus, for example, may have caused the initial loss of synchronisation, but we assume that there are no further faults during the execution of our model.

3.2 The System Model

The main components of the system model are a bus automaton, modeling a broadcast bus, and a number of station automata, modeling the computer nodes, which

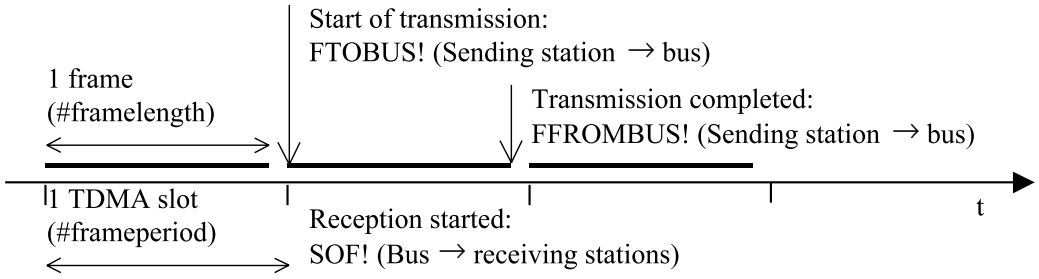


Figure 5: Successful Transmission and Reception.

send and receive messages on the bus. In the following we will briefly describe these automata. To indicate the type of identifiers, we use location, `variable`, #constant and ACTION. Table 2 in Appendix A gives a key to the variables, constants and actions used.

3.2.1 The Bus Automaton

The bus automaton, shown in Figure 4, models the behaviour of the system bus. When a message is broadcast, the bus automaton receives the input action FTOBUS from the sending station (sender), to indicate that the start of a frame is transmitted. This puts the bus automaton in location send in which it offers synchronisation with any receiving station (receiver) on the SOF action, to model the reception of the start of frame field in the receivers. At the end of a frame, the bus synchronises with the sender on the FFROMBUS action and changes control back to location idle. The whole scenario of a successfully transmitted frame on the bus is depicted in Figure 5.

If a collision occurs, the transmitting senders after the first one synchronise with the bus on the JAMTOBUS action, see Figure 6. The JAM action is used to model the reception of corrupted messages in the receivers, either as the result of a bus collision or because the start of frame was missed. The end of the corrupted message is modelled by the JAMFROMBUS action.

To model the presence of a message on the communications bus, the bus automaton updates the variable `busid`. Its value is #noid if the bus is idle or if a collision has occurred, or #0, #1, ..., #nodemax depending on which station is sending a message. A receiver must see a valid `busid` at the end of a message reception; otherwise, the message is considered corrupted.

3.2.2 The station automaton

The station automaton models the protocol behaviour in each computer node. In Figure 9 the complete automaton is shown annotated with labels on the important edges corresponding to the operations described in Section 2.3.

To model clock drift, each guard containing a clock variable is transformed to an interval corresponding to the size of clock drift. This is done automatically by the UPPAAL tool. Because the model-checker investigates all possible scenarios, a

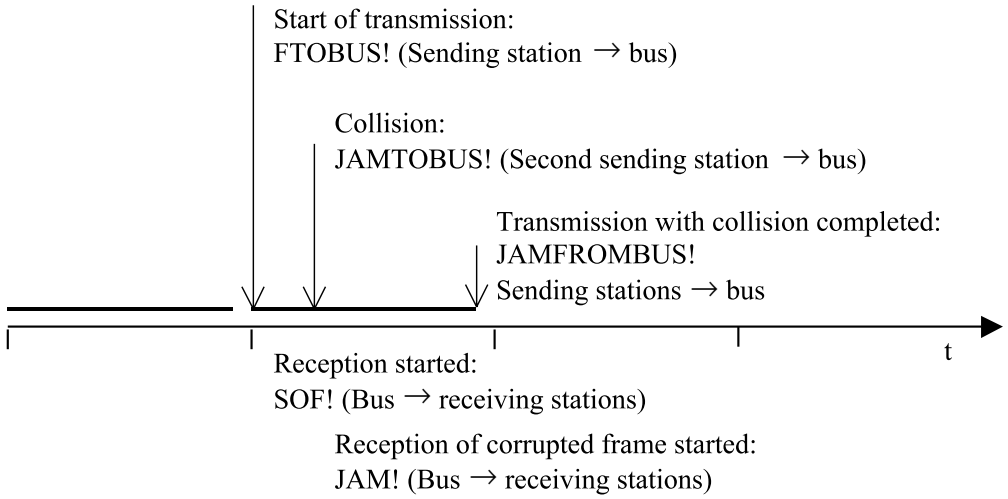


Figure 6: Message Transmission with Collision.

transition may be taken both early (fast clock) and late (slow clock) in the interval[1].

The automaton is divided in three areas, belonging to **N**ormal, re**S**ynchronisation and **R**ecover mode. Location names in the modes are identical where applicable, but end with **N**, **S**, and **R** to distinguish between them.

Before the protocol begins to execute, the variables `bitclock` and `idcount` are initialised to random values. The variable `bitclock` is initialised by allowing the automaton to take its first transition, from location start to openS, at any time in the interval $[0, \ldots, \#\text{frameper}]$, i.e. at any time in the TDMA slot. The variable `idcount` is initialised by an external initialisation automaton.

A station waits for bus activity in location `open`. It waits for #winsize time units in normal mode and for the duration of the entire TDMA slot in the two other modes. When a synchronisation with the bus is performed, `bitclock` and the counter `silence` are reset.

If a SOF action is received, the station enters location receiving for #framesize time units. If no collision has occurred after this time, the variable `busid` equals the id of the sending node and `idcount` is assigned this value. If a JAM synchronisation occurs or an initiated message reception fails, variable `idcount` is temporarily left unchanged.

If it is not possible to receive a message, `bitclock` is reset and `idcount` is incremented at the end of the TDMA slot. The error counter `errcount` is also incremented and, if no bus activity has occurred, `silence` is incremented.

When the various counters have been incremented at the end of a TDMA slot, the station may change mode. Mode changes to normal mode are made when counter `errcount` falls to #errmax (see 1d in Figure 9). In resynchronisation mode, a mode change to recover mode is done when counter `silence` reaches #silencetime (see 2e in Figure 9). Unless it is possible to reach normal mode, the station goes back to resynchronisation mode after the transmission of one frame.

[1]The technique is presented in [OSY94, DY95]

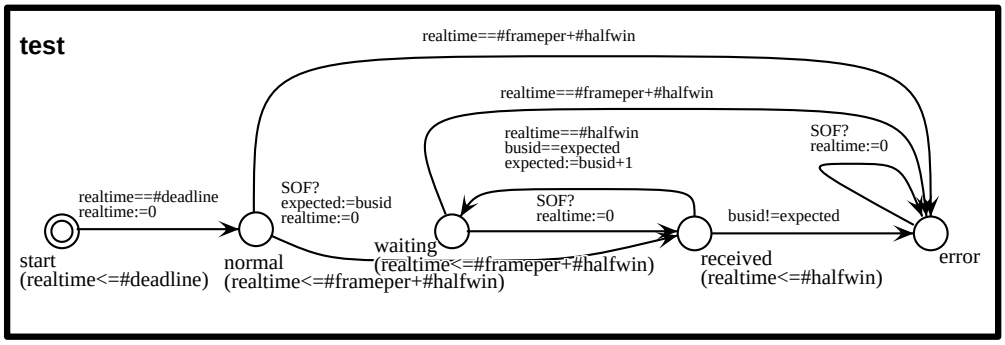


Figure 7: Test Automaton Used to Express Correctness Properties.

At the end of the TDMA slot, decision on message transmission is made as well. In normal mode, a station enters location `send` if counter `idcount` indicates that it owns the next TDMA slot. In recover mode, location `send` is entered only if counter `silence` is zero as well. This guarantees that a message is not sent in the first TDMA slot in recover mode, since `silence` is not reset until after the first slot.

4 Verification

A system consisting of four stations connected to a communication bus is modeled to analyse the behaviour of the DACAPO protocol. Each station is modeled as an instance of the station automaton (i.e. gci_0, gci_1, etc.). Station 3 is not permitted to broadcast in recover mode (it is not part of the recovery set), since this could cause repeated collisions. The bus automaton is used to model the communication bus. In addition, a test automaton is included to support verification.

To validate and verify properties of the protocol model, we use the UPPAAL toolbox [LPY97]. The model-checker in UPPAAL allows for verifications of invariant and bounded-liveness properties of networks of timed automata. An *invariant property* is in the form "p is always true" and may be used to ensure that certain unexpected situations never occur, e.g. "automaton A will never reach location bad". A *bounded-liveness property* is in the form "p is guaranteed to hold within time t" and may be used to check that an expected situation occurs within a specified time bound. This property can be checked by including the state of a test automaton in the invariant expression [JLS96, ABL98]. For example, if automaton test transits from location initial to expired at time t, the bounded-liveness property would be "automaton test in location expired implies that automaton A is in location ready".

4.1 Correctness Properties

The main correctness property of the start-up protocol requires all stations to enter normal mode within a bounded time (#deadline time units). To conveniently express this property, we introduce an auxiliary integer variable `n`. The value of `n` is incremented when a station enters normal mode (i.e. on the edges 2D and 3D in Figure 9) and decremented when it exits (i.e. on the edge 1D in Figure 9). A test automaton

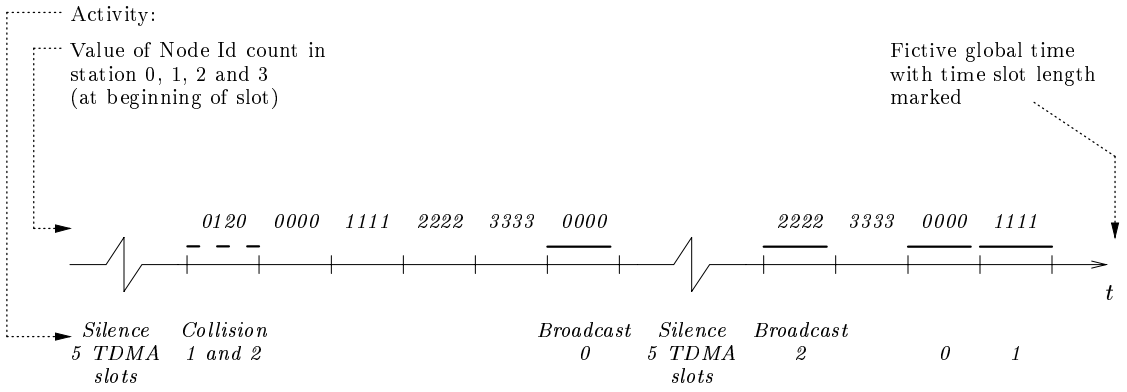


Figure 8: Worst Case Scenario.

that changes control from location start to normal at time #deadline is also included, see Figure 7.

Property (1) requires all stations to be in normal mode when the test automaton has left location start, i.e. at time #deadline:

$$\mathsf{INV}(\ (\ \neg \mathrm{test.start}\ ) \Rightarrow (\ \mathtt{n} = 4\ )\ ) \tag{1}$$

We also verify that the protocol operates correctly in normal mode. To do this, the test automaton has a location error that is entered unless there is a bus transmission once every TDMA slot, and the broadcast order is 0, 1, 2, 3, 0, 1, 2, 3, etc. Property (2) expresses the correctness property that automaton test never reaches location error.

$$\mathsf{INV}(\ \neg \mathrm{test.error}\ ) \tag{2}$$

UPPAAL verifies that both these properties hold in the system model. Moreover, the tool checks that the model never reaches a deadlock, a check that is necessary to make property (2) meaningful. Otherwise, it could have been the deadlock, rather than correct operation of the protocol model, that prevented the test automaton from reaching test.error.

Table 1 summarises the resource requirements of UPPAAL[2] installed on a Pentium Pro 200 MHz equipped with 256 Mb of memory.

4.2 Duration of Start-Up

An upper bound on #deadline can be established by iterating the analysis in UPPAAL with increasing #deadline until property (1) and (2) are satisfied. The worst case scenario (the largest #deadline where property (1) does not hold) occurs when there is a collision between two nodes during the first transmission attempt, see Figure 8. After the collision, the duration of the TDMA slot is longer than nominal for nodes 2 and 3. This is because, in the worst case, a bit clock synchronisation may occur

[2]The current UPPAAL version is 2.18.4, August 1998.

	Memory usage	Execution Time
3 stations, no clock drift	4 Mb	10 s
3 stations, clock drift	6 Mb	20 s
4 stations, no clock drift	65 Mb	691 s
4 stations, clock drift	253 Mb	2589 s

Table 1: Resource Usage for Verification.

at the end of the nominal TDMA slot when a corrupted message is present. Time is thus set back an entire TDMA slot, and the counter Node Id Count of these stations are not incremented at the normal time. The total delay until all nodes are in normal mode corresponds to about 21 TDMA slots.

5 Conclusions

We have formally described, in the model of timed automata, the start-up algorithm of a TDMA protocol for distributed systems with a broadcast bus. The protocol model of the start-up algorithm is analysed for a system consisting of four stations using the symbolic model-checking tool UPPAAL. As verification results we have shown that the protocol model becomes operational after an initial start-up phase which is bounded to, in the worst-case, 21 TDMA time slots. The worst-case scenario occurs if there is a collision on the first transmission attempt.

The analysis presented in this paper applies to systems with four stations, or less. Since many real system will contain more stations, this work will have to be extended. However, even the four station model is large compared with other examples verified using symbolic model-checking techniques. Consequently, an extension to five or more nodes will be very challenging.

As regards the protocol functionality, the DACAPO protocol uses ordinary messages in the start-up phase and distributes data during this period. Although it may take several TDMA slots before all nodes have reached normal mode, this therefore does not mean that the communication service is unavailable for the duration of that period. This can be compared to the Time Triggered Protocol (TTP), where a Blackout monitoring mode is entered when synchronisation is lost. In Blackout monitoring mode, special frames containing current time and system status are sent, but user data is not distributed. Also, if a node loses synchronisation while the rest of the ensemble continues normal operation, it must wait for a *re-integration frame* to be broadcast. These are only sent periodically, as defined by the applications programmer. Including system time in each frame as in DACAPO does cause some overhead, but single-node recovery can be done very rapidly and re-integration frames can be avoided altogether.

References

[ABL98] Luca Aceto, Augusto Bergueno, and Kim G. Larsen. Model Checking via Reachability Testing for Timed Automata. In Bernard Steffen, editor, *Proc. of the 4*th *Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, number 1384 in Lecture Notes in Computer Science, pages 263–280. Springer–Verlag, 1998.

[AD90] Rajeev Alur and David Dill. Automata for Modelling Real-Time Systems. In *Proc. of Int. Colloquium on Algorithms, Languages and Programming*, number 443 in Lecture Notes in Computer Science, pages 322–335, July 1990.

[BD87] Babaglou and Drummond. (Almost) no cost clock synchronisation. In *Proc. of the 17*th *IEEE Int. Symp. on Fault-Tolerant Computing*, pages 42–47, June 1987.

[DY95] C. Daws and S. Yovine. Two examples of verification of multirate timed automata with KRONOS. In *Proc. of the 16*th *IEEE Real-Time Systems Symposium*, pages 66–75. IEEE Computer Society Press, December 1995.

[JLS96] Henrik E. Jensen, Kim G. Larsen, and Arne Skou. Modelling and Analysis of a Collision Avoidance Protocol Using SPIN and UPPAAL. In *Proc. of 2*nd *Int. Workshop on the SPIN Verification System*, pages 1–20, August 1996.

[KG94] H. Kopetz and G. Grönsteidl. TTP - A Protocol for Fault-Tolerant Real-Time Systems. *IEEE Computer*, pages 14–23, January 1994.

[Kop93] H. Kopetz. Should Responsive Systems be Event-Triggered or Time Triggered? *EICE Trans. on Information and systems*, E76-D(11):1325–32, November 1993.

[KU95] P.J. Koopman and B.P. Upender. Time Division Multiple Access Without a Bus Master. Technical Report RR-9500470, United Technologies Research Center, 1995.

[LPY97] Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a Nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1(1–2):134–152, October 1997.

[LS94] Henrik Lönn and R. Snedsböl. Efficient synchronization, atomic broadcast and membership agreement in a TDMA protocol. In *Proc. of ISCA Int. Conf. on Parallel and Distributed Computing Systems*, 1994.

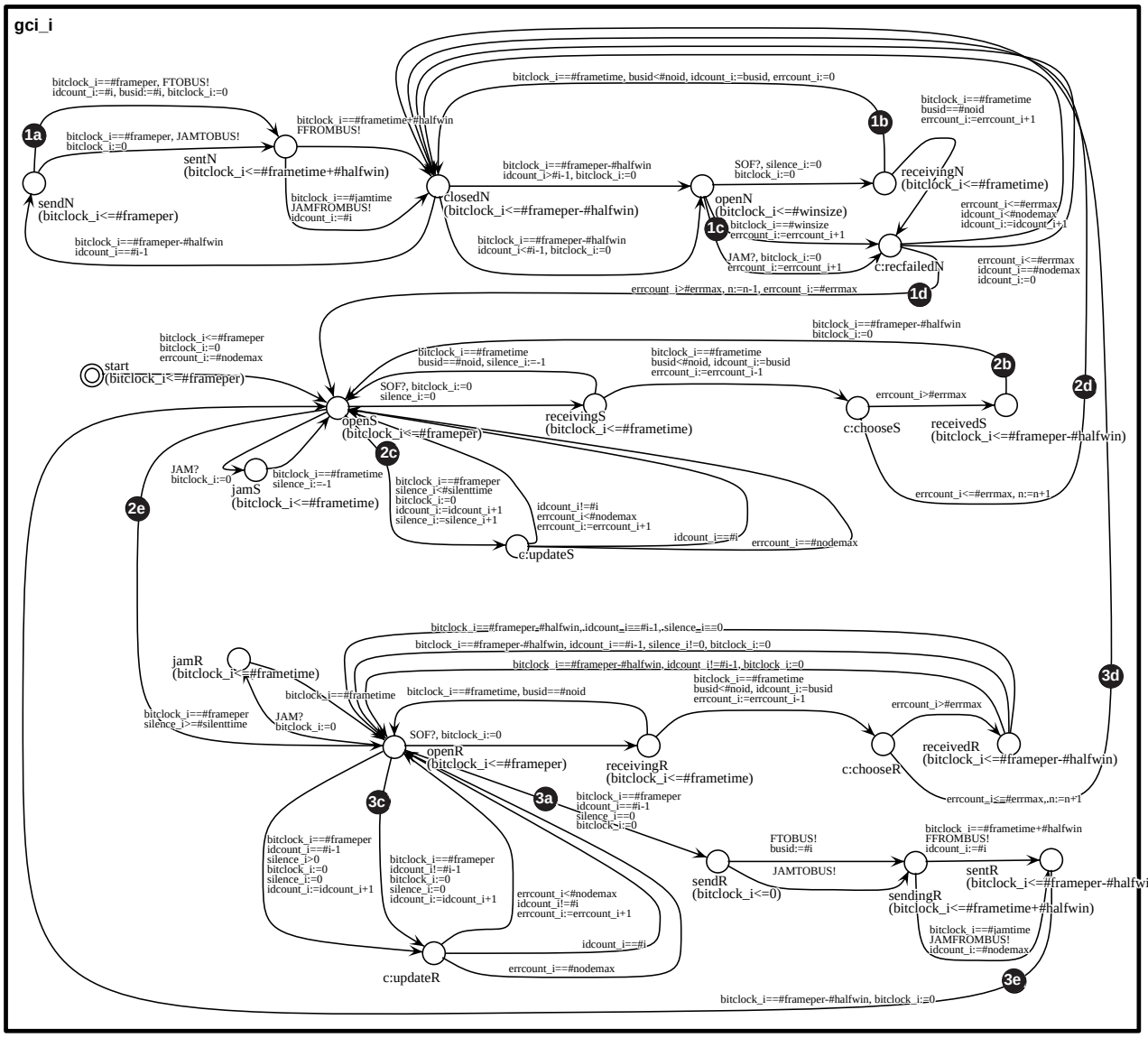
[LS95] H. Lönn and R. Snedsböl. Synchronisation in Safety-Critical Distributed Control Systems. In *Proc. of Int. Conf. on Architectures and Algorithms for Parallel Processing*, 1995.

[OSY94] A. Olivero, J. Sifakis, and S. Yovine. Using Abstractions for the Verification of Linear Hybrids Systems. In *Proc. of the 6*th *Int. Conf. on Computer Aided Verification*, number 818 in Lecture Notes in Computer Science, 1994.

[RLST95] B. Rostamzadeh, H. Lönn, R. Snedsböl, and J. Torin. DACAPO: A Distributed Computer Archiecture for Safety-Critical Control Applications. In *Proc. of IEEE Int. Symp. on Intelligent Vehicles*, 1995.

[Tor92] Jan Torin. Dependability in Complex Automotive Systems. Requirement Directions and Drivers. In *Proc. of Workshop on Safety and Reliability Engineering of Future Prometheus System*, 1992.

A Appendix



Note: Calculations involving idcount_i are made modulo #nodeant.

Figure 9: Timed Automaton Describing a Station.

Clock Variables	
`bitclock_i`	The local clock source for station #i
`Busclock`	The bus clock source. Used to enforce immediate delivery of messages
Integer Variables	
`idcount_i`	node id of the owner of the current TDMA time slot.
`silence_i`	number of TDMA time slots without bus activity.
`errcount_i`	number of empty or erroneous messages. Increased on TDMA slots with empty or erroneous messages.
`busid`	node id in the message on the bus.
`n`	Number of stations in normal mode. Used for invariant expressions during verification.
Actions	
SOF	Start Of Frame indication. Used to synchronise bit clock and start message reception
JAM	Bus activity indication. Used to synchronise bit clock.
FTOBUS	Indicates that a frame is broadcast to the bus.
FFROMBUS	Indicates that a frame is removed from the bus.
JAMTOBUS	Indicates that a frame is broadcast to the bus while it is busy (Resulting in a corrupted frame)
JAMFROMBUS	Indicates that a corrupted frame is removed from the bus.
Constants	
#frameper	Time between start of a TDMA slot, 224.
#frametime	Length of a message frame, 218.
#jamtime	Maximum duration of a tranmission of a corrupted message, 219.
#nodeant	Number of nodes in the system, 4.
#nodemax	The largest node id used (#nodeant-1) 3.
#winsize	The size of the reception window, 2.
#halfwin	The midpoint of the reception window (#winsize/2), 1.
#noid	Bus id used to indicate an idle bus, 99.
#errmax	Maximum number of errors tolerated in Normal mode ($\lfloor(\#\text{nodeant} - 1)/2\rfloor$), 1.
#silenttime	Number of silent TDMA slots before entering Recover Mode -1 (#nodeant-1), 3.
#i	Local node id, in the interval [0...#nodemax].

Table 2: Clock Variables, Integer Variables and Constants Used in the Automata.

Paper G

Magnus Lindahl, Paul Pettersson, and Wang Yi. Formal Design and Analysis of a Gear Controller. In Bernhard Steffen, editor, *Proceedings of the 4*th *Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, number 1384 in Lecture Notes in Computer Science, pages 281–297. Springer–Verlag 1998.

Formal Design and Analysis of a Gear Controller

Magnus Lindahl*, Paul Pettersson† and Wang Yi†

* Mecel AB, Göteborg, Sweden. Email: magnus.lindahl@mecel.se.

† Department of Computer Systems, Uppsala University, Sweden.
Email: {paupet,yi}@docs.uu.se.

Abstract. In this paper, we report on an application of the validation and verification tool kit UPPAAL in the design and analysis of a prototype gear controller, carried out in a joint project between industry and academia. We give a detailed description of the formal model of the gear controller and its surrounding environment, and its correctness formalised according to the informal requirements delivered by our industrial partner of the project. The second contribution of this paper is a solution to the problem we met in this case study, namely how to use a tool like UPPAAL, which only provides reachability analysis to verify bounded response time properties. The advantage of our solution is that we need no additional implementation work to extend the existing model-checker, but simple manual syntactical manipulation on the system description.

1 Introduction

Over the past few years, a number of modeling and verification tools for real-time systems [HHWT95, DOTY95, BLL+96] have been developed based on the theory of timed automata [AD94]. They have been successfully applied in various case-studies [BGK+96, JLS96, SMF97]. However, the tools have been mainly used in the academic community, namely by the tool developers. It has been a challenge to apply these tools to real-sized industrial case-studies. In this paper we report on an application of the verification tool-kit UPPAAL to a prototype gearbox controller developed in a joint project between industry and academia. The project has been carried out in collaboration between Mecel AB and Uppsala University.

The gearbox controller is a component in the real-time embedded system that operates in a modern vehicle. The gear-requests from the driver are delivered over a communication network to the gearbox controller. The controller implements the actual gear change by actuating the lower level components of the system, such as the clutch, the engine and the gearbox. Obviously, the behavior of the gearbox controller is critical to the safety of the vehicle. Simulation and testing have been the traditional ways to ensure that the behavior of the controller satisfies certain safety requirements. However these methods are by no means complete in finding errors though they are useful and practical. As a complement, formal techniques have been a promising approach to ensuring the correctness of embedded systems. The project is to use formal modeling techniques in the early design stages to describe design sketches, and to use symbolic simulators and model checkers as debugging and verification tools to ensure that the predicted behavior of the designed controller at each design phase, satisfies certain requirements under given assumptions on the environment where the gearbox controller is supposed to operate. The requirements on the controller and assumptions on the environment have been described by Mecel AB in an informal

document, and then formalised in the UPPAAL model and a simple linear-time logic based on the UPPAAL logic to deduce the design of the gearbox controller.

We shall give a detailed description of the formal model of the gearbox controller and its surrounding environment, and its correctness according to the informal requirements delivered by Mecel AB. Another contribution of this paper is a lesson we learnt in this case study, namely how to use a tool like UPPAAL, which only provides reachability analysis to verify bounded response time properties e.g. *if f_1 (a request) becomes true at a certain time point, f_2 (a response) must be guaranteed to be true within a time bound.* We present a logic and a method to characterise and model-check response time properties. The advantage of this approach is that we need no additional implementation work to extend the existing model-checker, but simple manual syntactical manipulation on the system description.

UPPAAL[1] is a tool suite for validation and symbolic model-checking of real-time systems. It consists of a number of tools including a graphical editor for system descriptions (based on Autograph), a graphical simulator, and a symbolic model–checker. In the design phase the symbolic simulator of UPPAAL is applied intensively to validate the dynamic behavior of each design sketch, in particular for fault detection, derivation of time constraints (e.g. the time bounds for which a gear change is guaranteed) and later also for debugging using diagnostic traces (i.e. counter examples) generated by the model-checker. The correctness of the gearbox controller design has been established by automatic proofs of 47 logical formulas derived from the informal requirements specified by Mecel AB. The verification was performed in a few seconds on a Pentium PC [2] running UPPAAL version 2.12.2.

The paper is organised as follows: In the next section we present a simple logic to characterise safety and response time properties and a method to model-check such properties. In section 4 and 5 the gearbox controller system and its requirements are informally and formally described. In section 6 the formal description of the system and its requirements are transformed using the technique developed in section 3 for verification by reachability analysis. Section 7 concludes the paper. Finally, in an appendix, we enclose the formal description of the whole system.

2 A Logic for Safety and Bounded Response Time Properties

At the start of the project, we found that it was not so obvious how to formalise (in the UPPAAL logic) the pages of informal requirements delivered by the design engineers. One of the reasons was that our logic is too simple, which can express essentially only invariant properties. It later became obvious that these requirements could be described in a simple logic, which can be model-checked by reachability analysis in combination with a certain syntactical manipulation on the model of the system to be verified. We also noticed that though the logic is so simple, it characterises the class

[1] Further information on installation and documentation for UPPAAL is available at `http://www.docs.uu.se/uppaal/`.

[2] 2.99 seconds on a Pentium 75MHz equipped with 24 MB of primary memory.

of logical properties verified in all previous case studies where UPPAAL is applied (see e.g. [BGK+96, JLS96]).

2.1 Timed Transition Systems and Timed Traces

A timed transition system is a labeled transition system with two types of labels: atomic actions and delay actions (i.e. positive reals), representing discrete and continuous changes of real-time systems.

Let $\mathcal{A}$ be a finite set of actions and $\mathcal{P}$ be a set of atomic propositions. We use $\mathbb{R}_+$ to stand for the set of non-negative real numbers, Δ for the set of delay actions $\{\epsilon(d) \mid d \in \mathbb{R}_+\}$, and Σ for the union $\mathcal{A} \cup \Delta$ ranged over by $\alpha, \alpha_1, \alpha_2$ etc.

Definition 2.1 (Timed Transition System) *A timed transition system over $\mathcal{A}$ and $\mathcal{P}$ is a tuple $\mathcal{S} = \langle S, s_0, \Leftrightarrow\!\!\rightarrow, V\rangle$, where S is a set of states, s_0 is the initial state, $\Leftrightarrow\!\!\rightarrow \subseteq S \times \Sigma \times S$ is a transition relation, and $V : S \to 2^{\mathcal{P}}$ is a proposition assignment function.* □

A trace σ of a timed transition system is an *infinite* sequence of transitions in the form:

$$\sigma = s_0 \overset{\alpha_0}{\Leftrightarrow\!\!\rightarrow} s_1 \overset{\alpha_1}{\Leftrightarrow\!\!\rightarrow} s_2 \overset{\alpha_2}{\Leftrightarrow\!\!\rightarrow} \ldots s_n \overset{\alpha_n}{\Leftrightarrow\!\!\rightarrow} s_{n+1} \ldots$$

where $\alpha_i \in \Sigma$. A position π of σ is a natural number. We use $\sigma[\pi]$ to stand for the πth state of σ, and $\sigma(\pi)$ for the πth transition of σ, i.e. $\sigma[\pi] = s_\pi$ and $\sigma(\pi) = s_\pi \overset{\alpha_\pi}{\Leftrightarrow\!\!\rightarrow} s_{\pi+1}$. We use $\delta(s \overset{\alpha}{\Leftrightarrow\!\!\rightarrow} s')$ to denote the duration of the transition, defined by $\delta(s \overset{\alpha}{\Leftrightarrow\!\!\rightarrow} s') = 0$ if $\alpha \in \mathcal{A}$ or d if $\alpha = \epsilon(d)$. Given positions i, k with $i \leq k$, we use $\mathcal{D}(\sigma, i, k)$ to stand for the accumulated delay of σ between the positions i, k, defined by $\mathcal{D}(\sigma, i, k) = \sum_{i \leq j < k} \delta(\sigma(j))$. We shall only consider *non-zeno* traces.

Definition 2.2 (Non-Zeno Traces) *A trace σ is non-zeno if for all natural number T there exists a position k such that $\mathcal{D}(\sigma, 0, k) > T$. For a timed transition system $\mathcal{S}$, we denote by $Tr(\mathcal{S})$ all non-zeno traces of $\mathcal{S}$ starting from the initial state s_0 of $\mathcal{S}$.* □

Note that the timed transition system defined above can also be represented finitely as a network of timed automata. For the definition of such networks, we refer to [LPY95, LPY97]. Let $\overline{A}$ be a network of timed automata with components $A_1 \ldots A_n$. We denote by $Tr(\overline{A})$ all non-zeno traces of the timed transition system $\overline{\mathcal{S}}$ i.e. $Tr(\overline{A}) = Tr(\overline{\mathcal{S}})$.

2.2 The Logic: Syntax and Semantics

The logic may be seen as a timed variant of a fragment of the linear temporal logic LTL, which does not allow nested applications of modal operators. It is to express invariant and bounded response time properties.

$$
\begin{array}{rcl}
(l,u) \models g & \textit{iff} & g(u) \\
(l,u) \models p & \textit{iff} & p \in V(l) \\
(l,u) \models \neg f & \textit{iff} & (l,u) \not\models f \\
(l,u) \models f_1 \wedge f_2 & \textit{iff} & (l,u) \models f_1 \text{ and } (l,u) \models f_2 \\
\sigma \models \mathsf{INV}(f) & \textit{iff} & \forall i : \sigma[i] \models f \\
\sigma \models f_1 \rightsquigarrow_{\leq T} f_2 & \textit{iff} & \forall i : (\sigma[i] \models f_1 \Rightarrow \exists k \geq i : (\sigma[k] \models f_2 \text{ and } \mathcal{D}(\sigma,i,k) \leq T))
\end{array}
$$

Table 1: Definition of Satisfiability.

Definition 2.3 (State-Formulas) *Assume that $\mathcal{C}$ is a set of clocks and P is a finite set of propositions. Let $\mathcal{F}_s$ denote the set of state-formulas over $\mathcal{C}$ and P ranged over by f, f_1, f_2 etc. defined as follows:*

$$f ::= g \mid p \mid \neg f \mid f_1 \wedge f_2$$

where $p \in P$ is an atomic proposition and g is an atomic clock constraint in the form $x \sim n$ or $x \Leftrightarrow y \sim n$ for $x, y \in \mathcal{C}$, $\sim \in \{<, \leq, =, \geq, >\}$ and n being a natural number. □

As usual, we use $f_1 \vee f_2$ to stand for $\neg(\neg f_1 \wedge \neg f_2)$, and tt and ff for $\neg f \vee f$ and $\neg f \wedge f$ respectively. Further, we use $f_1 \Rightarrow f_2$ to denote $\neg f_1 \vee f_2$.

Definition 2.4 (Trace-Formulas) *The set $\mathcal{F}_t$ ranged over by φ, ψ of trace-formulas over $\mathcal{F}_s$ is defined as follows:*

$$\varphi ::= \mathsf{INV}(f) \mid f_1 \rightsquigarrow_{\leq T} f_2$$

where T is a natural number. If f_1 and f_2 are boolean combinations of atomic propositions, we call $f_1 \rightsquigarrow_{\leq T} f_2$ a bounded response time formula. □

$\mathsf{INV}(f)$ states that f is an invariant property. A system satisfies $\mathsf{INV}(f)$ if all its reachable states satisfy f. It is useful to express safety properties, that is, *bad* things (e.g. deadlocks) should never happen, in other words, the system should always behave safely. $f_1 \rightsquigarrow_{\leq T} f_2$ is similar to the strong Until-operator in LTL, but with an explicit time bound. In addition to the time bound, it is also an invariant formula. It means that as soon as f_1 is true of a state, f_2 *must* be true within T time units. However it is not necessary that f_1 must be true continuously before f_2 becomes true as required by the traditional Until-operator.

We shall call a formula of the form $f_1 \rightsquigarrow_{\leq T} f_2$ a *bounded response time formula.* Intuitively, f_1 may be considered as a *request* and f_2 as a *response*; thus $f_1 \rightsquigarrow_{\leq T} f_2$ specifies the bound for the response time to be T.

We interpret $\mathcal{F}_s$ and $\mathcal{F}_t$ in terms of states and (infinite and non-zeno) traces of timed automata. We write $(l,u) \models f$ to denote that the state (l,u) satisfies the state-formula f and $\sigma \models \varphi$ to denote that the trace σ satisfies the trace-formula φ. The

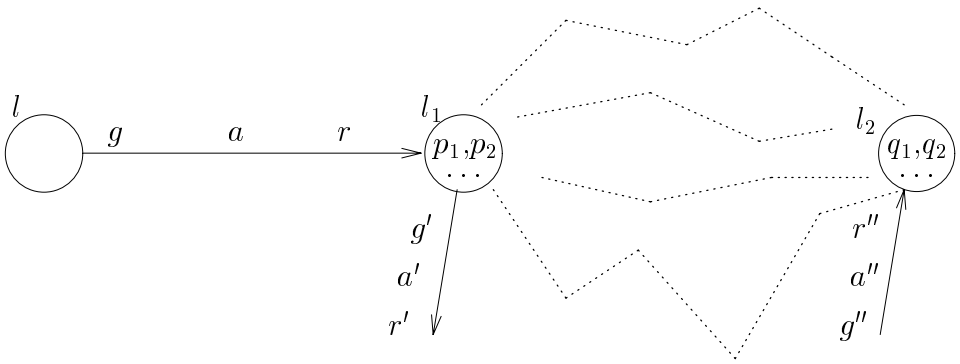


Figure 1: Illustration of a timed automaton A.

interpretation is defined on the structure of f and φ, given in Table 1. Naturally, if all the traces of a timed automaton satisfy a trace-formula, we say that the automaton satisfies the formula.

Definition 2.5 *Assume a network of automata $\overline{A}$ and a trace-formula φ. We write $\overline{A} \models \varphi$ iff $\sigma \models \varphi$ for all $\sigma \in Tr(\overline{A})$.* □

3 Verifying Bounded Response Time Properties by Reachability Analysis

The current version of UPPAAL[3] can only model-check invariant properties by reachability analysis. The question is how to use a tool like UPPAAL to check for bounded response time properties i.e. how to transform the model-checking problem $A \models f_1 \rightsquigarrow_{\leq T} f_2$ to a reachability problem. A standard solution is to translate the formula to a testing automaton t (see e.g. [JLS96]) and then check whether the parallel system $A||t$ can reach a designated state of t.

We take a different approach. We modify (or rather decorate) the automaton A according to the state-formulas f_1 and f_2, and the time bound T and then construct a state-formula f such that

$$\mathcal{M}(A) \models \mathsf{INV}(f) \text{ iff } A \models f_1 \rightsquigarrow_{\leq T} f_2$$

where $\mathcal{M}(A)$ is the modified version of A.

We study an example. As usual, assume that each node of an automaton is assigned implicitly a proposition $\mathsf{at}(l)$ meaning that the current control node is l. Consider an automaton A illustrated in Figure 1 and a formula $\mathsf{at}(l_1) \rightsquigarrow_{\leq 3} \mathsf{at}(l_2)$ (i.e. it should always reach l_2 from l_1 within 3 time units). To check whether A satisfies the formula, we introduce an extra clock $c \in \mathcal{C}$ and a boolean variable[4] v_1 into the automaton A, that should be initiated with ff. Assume that the node l_1 has no local loops, i.e. containing no edges leaving and entering l_1. We modify the automaton A as follows:

[3]The current version of UPPAAL is 2.12.2.

[4]Note that a boolean variable may be represented by an integer variable in UPPAAL.

1. Duplicate all edges entering node l_1.
2. Add $\neg v_1$ as a guard to the original edges entering l_1.
3. Add $v_1 := \mathtt{tt}$ and $c := 0$ as reset-operations to the original edges entering l_1.
4. Add v_1 as a guard to the auxiliary copies of the edges entering l_1.
5. Add $v_1 := \mathtt{ff}$ as a reset-operation to all the edges entering l_2.

The modified (decorated) automaton $\mathcal{M}(A)$ is illustrated in Figure 2. Now, we claim that

$$\mathcal{M}(A) \models \mathsf{INV}(v_1 \Rightarrow c \leq 3) \;\text{ iff }\; A \models \mathsf{at}(l_1) \leadsto_{\leq 3} \mathsf{at}(l_2)$$

The invariant property $v_1 \Rightarrow c \leq 3$ states that either $\neg v_1$ or if v_1 then $c \leq 3$. There is only one situation that violates the invariant: v_1 and $c > 3$. Due to the progress property of time (or non-zenoness), the value of c should always increase. It will sooner or later pass 3. But if l_2 is reached before c reaches 3, v_1 will become $\mathtt{ff}$. Therefore, the only way to keep the invariant property true is that l_2 is reached within 3 time units whenever l_1 is reached.

The above method may be generalised to efficiently model-check response time formulas for networks of automata. Let $\boldsymbol{\mathcal{A}}(f)$ denote the set of atomic propositions occurring in a state-formula f. Assume a network $\overline{A}$ and a response time formula $f_1 \leadsto_{\leq T} f_2$. For simplicity, we consider the case when only atomic propositions occur in f_1 and f_2. Note that this is not a restriction, the result can be easily extended to the general case which also allows clock constraints in f_1 and f_2. We introduce to $\overline{A}$ the following auxiliary variables:

1. an auxiliary clock $c \in \mathcal{C}$ and an boolean variable v_1 (to denote the truth value of f_1), and
2. an auxiliary boolean variable v_p for all $p \in \boldsymbol{\mathcal{A}}(f_1) \cup \boldsymbol{\mathcal{A}}(f_2)$.

Assume that all the booleans of $\boldsymbol{\mathcal{A}}(f_1), \boldsymbol{\mathcal{A}}(f_2)$ and v_1 are initiated to $\mathtt{ff}$.

Let $\mathcal{E}(f)$ denote the boolean expression by replacing all $p \in \boldsymbol{\mathcal{A}}(f)$ with their corresponding boolean variable v_p. As usual, $\mathcal{E}(f)[\mathtt{tt}/v_p]$ denotes a substitution that replaces v_p with $\mathtt{tt}$ in $\mathcal{E}(f)$. This can be extended in the usual way to set of substitutions. For instance, the truth value of f at a given state s may be calculated by $\mathcal{E}(f)[\mathtt{tt}/v_p | p \in V(s)][\mathtt{ff}/v_p | p \notin V(s)]$.

Now we are ready to construct a decorated version $\mathcal{M}(\overline{A})$ for the network $\overline{A}$. We modify all the components A_i of $\overline{A}$ as follows:

1. For all edges of A_i, entering a node l_1 such that $V(l_1) \cap \boldsymbol{\mathcal{A}}(f_1) \neq \emptyset$:
   (a) Make two copies of each such edge.
   (b) To the original edge, add v_1 as a guard.
   (c) To the first copy, add $\neg v_1 \wedge \mathcal{E}(f_1)[\mathtt{tt}/v_p | p \in V(l_1)]$ as a guard and $c := 0, v_1 := \mathtt{tt}$ and $v_p := \mathtt{tt}$ for all $p \in V(l_1)$ as reset-operations.
   (d) To the second copy, add $\neg v_1 \wedge \neg \mathcal{E}(f_1)[\mathtt{tt}/v_p | p \in V(l_1)]$ as a guard and $v_p := \mathtt{tt}$ for all $p \in V(l_1)$ as reset-operations.
2. For all edges of A_i leaving a node l_1 such that $V(l_1) \cap \boldsymbol{\mathcal{A}}(f_1) \neq \emptyset$: add $v_p := \mathtt{ff}$ for all $p \in V(l_1)$ as reset-operations.

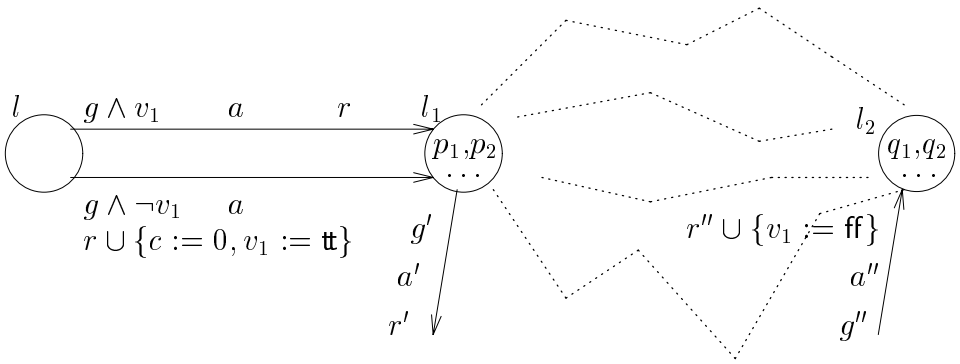


Figure 2: Illustration of a modified timed automaton $\mathcal{M}(A)$ of A.

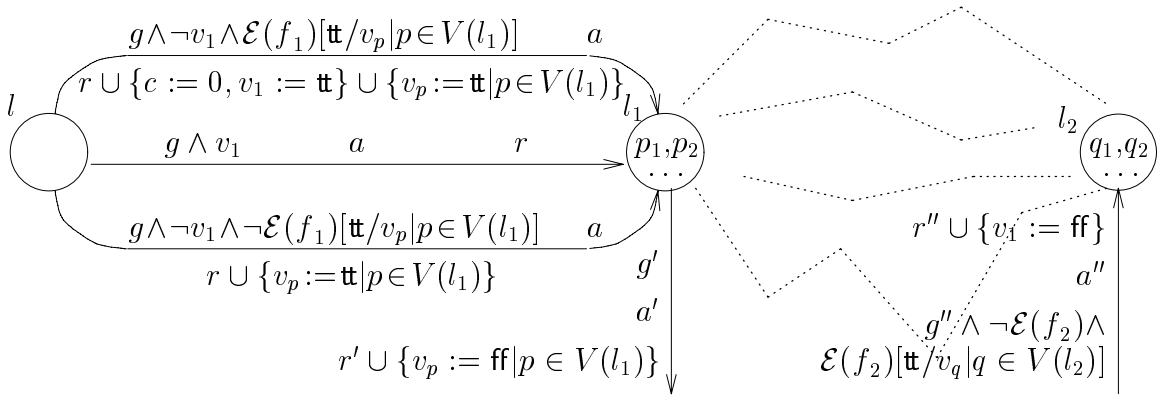


Figure 3: Illustration of a modified timed automaton $\mathcal{M}(A_i)$ of A_i.

3. For all edges of A_i entering a node l_2 such that $V(l_2) \cap \boldsymbol{\mathcal{A}}(f_2) \neq \emptyset$: add $\neg\mathcal{E}(f_2) \wedge \mathcal{E}(f_2)[\mathsf{tt}/v_q | q \in V(l_2)]$ as a guard and $v_1 := \mathsf{ff}$ as a reset-operation.
4. Finally, remove $v_p := \mathsf{tt}$ and $v_p := \mathsf{ff}$ whenever they occur at the same edge[5].

Thus, we have a decorated version $\mathcal{M}(A_i)$ for each A_i of $\overline{A}$. Assume that a component automaton A_i is as illustrated in Figure 1; its decorated version $\mathcal{M}(A_i)$ is shown in Figure 3. We shall take $\mathcal{M}(A_1) \,||\, \ldots \,||\, \mathcal{M}(A_n)$ to be the decorated version of $\overline{A}$, i.e. $\mathcal{M}(\overline{A}) \equiv \mathcal{M}(A_1) \,||\, \ldots \,||\, \mathcal{M}(A_n)$. For a bounded response time formula $f_1 \rightsquigarrow_{\leq T} f_2$, we now have

$$\mathcal{M}(\overline{A}) \models \mathsf{INV}(v_1 \Rightarrow c \leq T) \;\text{ iff }\; \overline{A} \models f_1 \rightsquigarrow_{\leq T} f_2$$

Note that we could have constructed the product automaton of $\overline{A}$ first. Then the construction of $\mathcal{M}(\overline{A})$ from the product automaton would be much simpler. But the size of $\mathcal{M}(\overline{A})$ will be much larger; it will be exponential in the size of the component automata. Our construction here is purely syntactical based on the syntactical structure of each component automaton. The size of $\mathcal{M}(\overline{A})$ is in fact linear in the size of the component automata. It is particularly appropriate for a tool like UPPAAL, that is based on on-the-fly generation of the state-space of a network. For each component

[5]This means that a proposition p is assigned to both the source and the target nodes of the edge; v_p must have been assigned tt on all the edges entering the source node.

automaton A, the size of $\mathcal{M}(A)$ can be calculated precisely as follows: In addition to one auxiliary clock c and $|P(f_1) \cup P(f_2)|$ boolean variables in $\mathcal{M}(A)$, the number of edges of $\mathcal{M}(A)$ is $3 \times |\Leftrightarrow\!\!\rightarrow_A|$ where $|\Leftrightarrow\!\!\rightarrow_A|$ is the number of edges of A (note that no extra nodes introduced in $\mathcal{M}(A)$).

Note also that in the above construction, we have the restriction that f_1 and f_2 contain no constraints, but only atomic propositions. The construction can be easily generalised to allow constraints by considering each constraint as a proposition and decorating each location (that is, the incoming edges) where the constraint could become true when the location is reached. In fact, this is what we did above on the boolean expressions (constraints) $\mathcal{E}(f_1)$ and $\mathcal{E}(f_2)$.

4 The Gear Controller

In this section we informally describe the functionality and the requirements of the gear controller proposed by Mecel AB, as well as the abstract behavior of the environment where the controller is supposed to operate.

4.1 Functionality

The gear controller changes gears by requesting services provided by the components in its environment. The interaction with these components is over the vehicles communication network. A description of the gear controller and its interface is as follows.

Interface: The interface receives service requests and keeps information about the current status of the gear controller, which is either changing gear or idling. The user of this service is either the driver using the gear stick or a dedicated component implementing a gear change algorithm. The interface is assumed to respond when the service is completed.

Gear Controller: The only user of the gear controller is its interface. The controller performs a gear change in five steps beginning when a gear change request is received from the interface. The first step is to accomplish a zero torque transmission, making it possible to release the currently set gear. Secondly the gear is released. The controller then achieves synchronous speed over the transmission and sets the new gear. Once the gear is set the engine torque is increased so that the same wheel torque level as before the gear change is achieved.

Under difficult driving conditions the engine may not be able to accomplish zero torque or synchronous speed over the transmission. It is then possible to change gear using the clutch. By opening the clutch, and consequently the transmission, the connection between the engine and the wheels is broken. The gearbox is at this state able to release and set the new gear, as zero torque and synchronous speed is no longer required. When the clutch closes it safely bridges the speed and torque differences between the engine and the wheels. We refer to these exceptional cases as *recoverable errors*.

The environment of the gear controller consists of the following three components:

Gearbox: It is an electrically controlled gearbox with control electronics. It provides services to *set* a gear in 100 to 300 ms and to *release* a gear in 100 to 200 ms. If a setting or releasing-operation of a gear takes more than 300 ms or 200 ms respectively, the gearbox will indicate this and stop in a specific error state.

Clutch: It is an electrically controlled clutch that has the same sort of basic services as the gearbox. The clutch can *open* or *close* within 100 to 150 ms. If a opening or closing is not accomplish within the time bounds, the clutch will indicate this and reach a specific error state.

Engine: The engine offers three modes of operation: normal torque, zero torque, and synchronous speed. The normal mode is *normal torque* where the engine gives the requested engine torque. In *zero torque* mode the engine will try to find a zero torque difference over the transmission. Similarly, in *synchronous speed* mode the engine searches zero speed difference between the engine and the wheels[6]. The maximum time bound searching for zero torque is limited to 400 ms within which a safe state is entered. Furthermore, the maximum time bound for synchronous speed control is limited to 500 ms. If 500 ms elapse the engine enters an error state.

We will refer the error states in the environment as *unrecoverable errors* since it is impossible for the gear controller alone to recover from these errors.

4.2 Requirements

In this section we list the informal requirements and desired functionality on the gear controller, provided by Mecel AB. The requirements are to ensure the correctness of the gear controller. A few operations, such as gear changes and error detections, are crucial to the correctness and must be guaranteed within certain time bounds. In addition, there are also requirements on the controller to ensure desired qualities of the vehicle, such as: good comfort, low fuel consumption, and low emission.

1. **Performance.** These requirements limit the maximum time to perform a gear change when no unrecoverable errors occur.
   (a) A gear change should be completed within 1.5 seconds.
   (b) A gear change, under normal operation conditions, should be performed within 1 second.
2. **Predictability**. The predictability requirements are to ensure strict synchronisation and control between components.
   (a) There should be no deadlocks in the system.
   (b) When the engine is regulating torque, the clutch should be closed.
   (c) The gear has to be set in the gearbox when the engine is regulating torque.

[6]Synchronous speed mode is used only when the clutch is open or no gear is set.

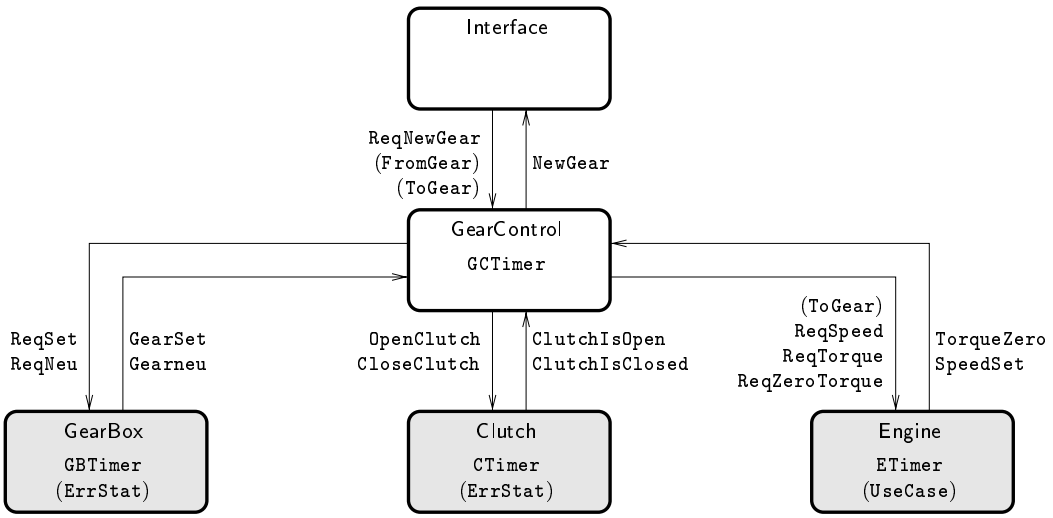


Figure 4: A Flow-Graph of the Gearbox System.

3. **Functionality.** The following requirements are to ensure the desired functionality of the gear controller.
   (a) It is able to use all gears.
   (b) It uses the engine to enhance zero torque and synchronous speed over the transmission.
   (c) It uses the gearbox to set and release gears.
   (d) It is allowed to use the clutch in difficult conditions.
   (e) It does not request zero torque when changing from neutral gear.
   (f) The gear controller does not request synchronous speed when changing to neutral gear.
4. **Error Detection.** The gear controller detects and indicates error only when:
   (a) the clutch is not opened in time,
   (b) the clutch is not closed in time,
   (c) the gearbox is not able to set a gear in time,
   (d) the gearbox is not able to release a gear in time.

5 Formal Description of the System

To design and analyse the gear controller we model the controller and its environment in the UPPAAL model [LPY97]. The modeling phase has been separated in two steps. First a model of the environment is created, as its behavior is specified in advance as assumptions (see Section 4.1). Secondly, the controller itself and its interface are designed to be functionally correct in the given environment. Figure 4 shows a flow-graph of the resulting model where nodes represent automata and edges represent synchronisation channels or shared variables (enclosed within parenthesis). The gear controller and its interface are modeled by the automata GearControl (GC)

and Interface (I). The environment is modeled by the three automata: Clutch (C), Engine (E), and GearBox (GB).

The system uses six variables. Four are timers that measure 1/1000 of seconds (ms): `GCTimer`, `GBTimer`, `CTimer` and `ETimer`. The two other variables, named `FromGear` and `ToGear`, are used at gear change requests[7]. In the following we describe the five automata of the system.

The three automata of the environment model the basic functionality and time behavior of the components in the environment. The components have two channels associated with each service: one for requests and one to respond when service have been performed.

Gearbox: In automaton GearBox, shown in Figure 5, inputs on channel `ReqSet` request a gear set and the corresponding response on `GearSet` is output if the gear is successfully set. Similarly, the channel `ReqNeu` requests the neutral gear and the response `GearNeu` signals if the gear is successfully released. If the gearbox fails to set or release a gear the locations named ErrorSet and ErrorNeu are entered respectively.

Clutch: The automaton Clutch is shown in Figure 8. Inputs on channels `OpenClutch` and `CloseClutch` instruct the clutch to open and close respectively. The corresponding response channels are `ClutchIsOpen` and `ClutchIsClosed`. If the clutch fails to open or close it enters the location ErrorOpen and ErrorClose respectively.

Engine: The automaton Engine, shown in Figure 6, accepts incoming requests for synchronous speed, a specified torque level or zero torque on the channels `ReqSpeed`, `ReqTorque` and `ReqZeroTorque` respectively. The actual torque level or requested speed is not modeled since it does not affect the design of the gear controller[8]. The engine responds on the channels `TorqueZero` and `SpeedSet` when the services have been completed. Requests for specific torque levels (i.e. signal `ReqTorque`) are not answered, instead torque is assumed to increase immediately after the request. If the engine fails to deliver zero torque or synchronous speed in time, it enters location CluthOpen without responding to the request. Similarly, the location ErrorSpeed is entered if the engine regulates on synchronous speed in too long time.

Given the formal model of the environment, the gear controller has been designed to satisfy both the functionality requirements given in Section 4.1, and the correctness requirements in Section 4.2

Gear Controller: The GearControl automaton is shown in Figure 7. Each main loop implements a gear change by interacting with the components of the environment. The designed controller measures response times from the components to detect errors (as failures are not signaled). The reaction of the controller

[7] The domains of `FromGear` and `ToGear` are bounded to $\{0, ..., 6\}$, where 1 to 5 represent gear 1 to gear 5, 0 represents gear N, and 6 is the reverse gear.

[8] Hence, the time bound for finding zero torque (i.e. 400 ms) should hold when decreasing from an arbitrary torque level.

depends on how serious the occurred error is. It either recovers the system from the error, or terminates in a pre-specified location that points out the (unrecoverable) error: COpenError, CCloseError, GNeuError or GSetError. Recoverable errors are detected in the locations CheckTorque and CheckSyncSpeed.

Interface: The automaton Interface shown in Figure 9, requests gears R, N, 1, ..., 5 from the gear controller. Requests and responses are sent through channel `ReqNewGear` and channel `NewGear` respectively. When a request is sent, the shared variables `FromGear` and `ToGear` are assigned values corresponding to the current and the requested new gear respectively.

6 Formal Validation and Verification

In this section we formalise the informal requirements given in Section 4.2 and prove their correctness using the symbolic model-checker of UPPAAL.

To enable formalisation (and verification) of requirements, we decorate the system description with two integer variables, `ErrStat` and `UseCase`. The variable `ErrStat` is assigned values at unrecoverable errors: 1 if Clutch fails to close, 2 if Clutch fails to open, 3 if GearBox fails to set a gear, and 4 if GearBox fails to release a gear. The variable `UseCase` is assigned values whenever a recoverable error occurs in Engine: 1 if it fail to deliver zero torque, and 2 if it is not able to find synchronous speed. The system model is also decorated to enable verification of bounded response time properties, as described in Section 3.

Before formalising the requirement specification of the gear controller we define negation and conjunction for the bounded response time operator and the invariant operator defined in Section 3,

$$\begin{aligned} \overline{A} \models \varphi_1 \wedge \varphi_2 \quad &\text{if and only if} \quad \overline{A} \models \varphi_1 \text{ and } \overline{A} \models \varphi_2 \\ \overline{A} \models \neg\varphi \quad &\text{if and only if} \quad \overline{A} \not\models \varphi \end{aligned}$$

We also extend the (implicit) proposition $\mathsf{at}(l)$ to $\mathsf{at}(A, l)$, meaning that the control location of automaton A is currently l. Finally, we introduce $\mathsf{Poss}(f)$ to denote $\neg\mathsf{INV}(\neg f)$, $f_1 \not\leadsto_{\leq T} f_2$ to denote $\neg(f_1 \leadsto_{\leq T} f_2)$, and $A.l$ to denote $\mathsf{at}(A, l)$. We are now ready to formalise the requirements.

6.1 Requirement Specification

The first performance requirement 1a, i.e. that a gear change must be completed within 1.5 seconds given that no unrecoverable errors occur, is specified in property 1. It requires the location GearChanged in automaton GearControl to be reached within 1.5 seconds after location Initiate has been entered. Only scenarios without unrecoverable errors are considered as the value of the variable `ErrStat` is specified to be zero[9]. To consider scenarios with normal operation we restrict also the value

[9]Recall that the variable `ErrStat` is assigned a positive value (i.e. greater than zero) whenever an unrecoverable error occurs.

$$\mathsf{GearControl.Initiate} \rightsquigarrow_{\leq 1500} ((\texttt{ErrStat} = 0) \Rightarrow \mathsf{GearControl.GearChanged}) \quad (1)$$

$$\mathsf{GearControl.Initiate} \rightsquigarrow_{\leq 1000} ((\texttt{ErrStat} = 0 \wedge \texttt{UseCase} = 0) \Rightarrow \mathsf{GearControl.GearChanged}) \quad (2)$$

$$\mathsf{Clutch.ErrorClose} \rightsquigarrow_{\leq 200} \mathsf{GearControl.CCloseError} \quad (3)$$

$$\mathsf{Clutch.ErrorOpen} \rightsquigarrow_{\leq 200} \mathsf{GearControl.COpenError} \quad (4)$$

$$\mathsf{GearBox.ErrorIdle} \rightsquigarrow_{\leq 350} \mathsf{GearControl.GSetError} \quad (5)$$

$$\mathsf{GearBox.ErrorNeu} \rightsquigarrow_{\leq 200} \mathsf{GearControl.GNeuError} \quad (6)$$

$$\mathsf{INV}\ (\ \mathsf{GearControl.CCloseError} \Rightarrow \mathsf{Clutch.ErrorClose}\ ) \quad (7)$$

$$\mathsf{INV}\ (\ \mathsf{GearControl.COpenError} \Rightarrow \mathsf{Clutch.ErrorOpen}\ ) \quad (8)$$

$$\mathsf{INV}\ (\ \mathsf{GearControl.GSetError} \Rightarrow \mathsf{GearBox.ErrorIdle}\ ) \quad (9)$$

$$\mathsf{INV}\ (\ \mathsf{GearControl.GNeuError} \Rightarrow \mathsf{GearBox.ErrorNeu}\ ) \quad (10)$$

$$\mathsf{INV}\ (\ \mathsf{Engine.ErrorSpeed} \Rightarrow \texttt{ErrStat} \neq 0\ ) \quad (11)$$

$$\bigwedge_{i \in \{R,N,1,\ldots,5\}} \mathsf{Poss}\ (\ \mathsf{Gear.Gear}_i) \quad (12)$$

$$\mathsf{INV}\ (\ \mathsf{Engine.Torque} \Rightarrow \mathsf{Clutch.Closed}\ ) \quad (13)$$

$$\bigwedge_{i \in \{R,1,\ldots,5\}} \mathsf{INV}((\ \mathsf{GearControl.Gear} \wedge \mathsf{Gear.Gear}_i) \Rightarrow \mathsf{Engine.Torque}) \quad (14)$$

Table 2: Requirement Specification

of variable `UseCase` to zero (i.e. no recoverable errors occurs). Property 2 requires gear changes to be completed within one second given that the system is operating normally.

The properties 3 to 6 require the system to terminate in known error-locations that point out the specific error when errors occur in the clutch or the gear (requirements 4a to 4d). Up to 350 ms is allowed to elapse between the occurrence of an error and that the error is indicated in the gear controller. The properties 7 to 10 restrict the controller design to indicate an error *only* when the corresponding error has arised in the components. Observe that no specific location in the gear controller is dedicated to indicate the unrecoverable error that may occur when the engines speed-regulation is interrupted (i.e. when location Engine.ErrorSpeed is reached). Property 11 requires that no such location is needed since this error is always a consequence of a preceding unrecoverable error in the clutch or in the gear.

Property 12 holds if the system is able to use all gears (requirement 3a). Furthermore, for full functionality and predictability, the system is required to be deadlock-free (requirement 2a).

The properties 13 and 14 specify the informal predictability requirements 2b and 2c.

A number of functionality requirements specify how the gear controller should interact with the environment (e.g. 3a to 3f). These requirements have been used

$$\mathsf{GearControl.Initiate} \rightsquigarrow_{<900} ((\texttt{ErrStat} = 0 \wedge \texttt{UseCase} = 0) \Rightarrow \mathsf{GearControl.GearChanged}) \quad (15)$$

$$\mathsf{GearControl.Initiate} \not\rightsquigarrow_{\leq 899} ((\texttt{ErrStat} = 0 \wedge \texttt{UseCase} = 0) \Rightarrow \mathsf{GearControl.GearChanged}) \quad (16)$$

Table 3: Time Bounds

to design the gear controller. They have later been validated using the simulator in UPPAAL and have not been formally specified and verified.

Time Bound Derivation

Property 1 requires that a gear change should be performed within one second. Even though this is an interesting property in itself one may ask for the *lowest* time bound for which a gear change is *guaranteed*. We show that the time bound is 900 ms for error-free scenarios by proving that the change is guaranteed at 900 ms (property 15), and that the change is possibly *not* completed at 899 ms (property 16). Similarly, for scenarios when the engine fails to deliver zero torque we derive the bound 1055 ms, and if synchronous speed is not delivered in the engine the time bound is 1205 ms.

We have shown the shortest time for which a gear change is *possible* in the three scenarios to be: 150 ms, 550 ms, and 450 ms. However, gear changes involving neutral gear may be faster as the gear does not have to be released (when changing from gear neutral) or set (when changing to gear neutral). Finally, we consider the same three scenarios but without involving neutral gear by constraining the values of the variables `FromGear` and `ToGear`. The derived time bounds are: 400 ms, 700 ms and 750.

Verification Results

We have verified totally 47 properties of the system[10] using UPPAAL installed on a 75 MHz Pentium PC equipped with 24 MB of primary memory. The verification of all the properties consumed 2.99 second.

7 Conclusion

In this paper, we have reported an industrial case study in applying formal techniques for the design and analysis of control systems for vehicles. The main output of the case-study is a formally described gear controller and a set of formal requirements. The designed controller has been validated and verified using the tool UPPAAL to satisfy the safety and functionality requirements on the controller, provided by Mecel

[10] A complete list of the verified properties can be found in the full version of this paper.

AB. It may be considered as one piece of evidence that the validation and verification tools of today are mature enough to be applied in industrial projects.

We have given a detailed description of the formal model of the gear controller and its surrounding environment, and its correctness formalised in 47 logical formulas according to the informal requirements delivered by industry. The verification was performed in a few seconds on a Pentium PC running UPPAAL version 2.12.2. Another contribution of this paper is a solution to a problem we got in this case study, namely how to use a tool like UPPAAL, which only provides reachability analysis to verify bounded response time properties. We have presented a logic and a method to characterise and model–check such properties by reachability analysis in combination with simple syntactical manipulation on the system description.

This work concerns only one component, namely gear controller of a control system for vehicles. Future work, naturally include modelling and verification of the whole control system. The project is still in progress. We hope to report more in the near future on the project.

Acknowledgements

The work has been supported by the ASTEC competence center (Advanced Software Technology) at Uppsala University. The authors want to thank Johan Bengtsson who developed a preliminary version of the UPPAAL model for the gear box system, and Hans Hansson and Mikael Strömberg for many fruitful discussions.

References

[AD94] R. Alur and D. Dill. Automata for Modelling Real-Time Systems. *Theoretical Computer Science*, 126(2):183–236, April 1994.

[BGK+96] Johan Bengtsson, W.O. David Griffioen, Kåre J. Kristoffersen, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. Verification of an Audio Protocol with Bus Collision Using UPPAAL. In Rajeev Alur and Thomas A. Henzinger, editors, *Proc. of the 8*th *Int. Conf. on Computer Aided Verification*, number 1102 in Lecture Notes in Computer Science, pages 244–256. Springer–Verlag, July 1996.

[BLL+96] Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. UPPAAL in 1995. In *Proc. of the 2*nd *Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, number 1055 in Lecture Notes in Computer Science, pages 431–434. Springer–Verlag, March 1996.

[DOTY95] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool KRONOS. In Rajeev Alur, Thomas A. Henzinger, and Eduardo D. Sontag, editors, *Proc. of Workshop on Verification and Control of Hybrid Systems III*, number 1066 in Lecture Notes in Computer Science, pages 208–219. Springer–Verlag, October 1995.

[HHWT95] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. HYTECH: The Next Generation. In *Proc. of the 16*th *IEEE Real-Time Systems Symposium*, pages 56–65. IEEE Computer Society Press, December 1995.

[JLS96] Henrik E. Jensen, Kim G. Larsen, and Arne Skou. Modelling and Analysis of a Collision Avoidance Protocol Using SPIN and UPPAAL. In *Proc. of 2*nd *Int. Workshop on the SPIN Verification System*, pages 1–20, August 1996.

[LPY95] Kim G. Larsen, Paul Pettersson, and Wang Yi. Compositional and Symbolic Model-Checking of Real-Time Systems. In *Proc. of the 16*th *IEEE Real-Time Systems Symposium*, pages 76–87. IEEE Computer Society Press, December 1995.

[LPY97] Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a Nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1(1–2):134–152, October 1997.

[SMF97] Thomas Stauner, Olaf Müller, and Max Fuchs. Using HyTech to Verify an Automotive Control System. In *Proc. Hybrid and Real-Time Systems, Grenoble, March 26-28, 1997*. Technische Universität München, Lecture Notes in Computer Science, Springer, 1997.

A The System Description

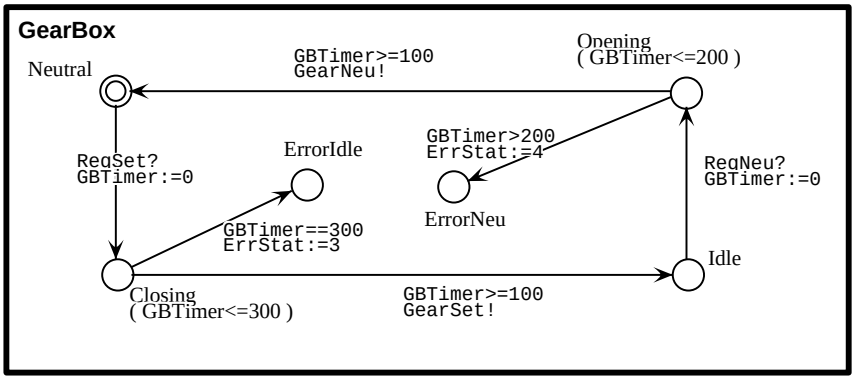


Figure 5: The Gearbox Automaton.

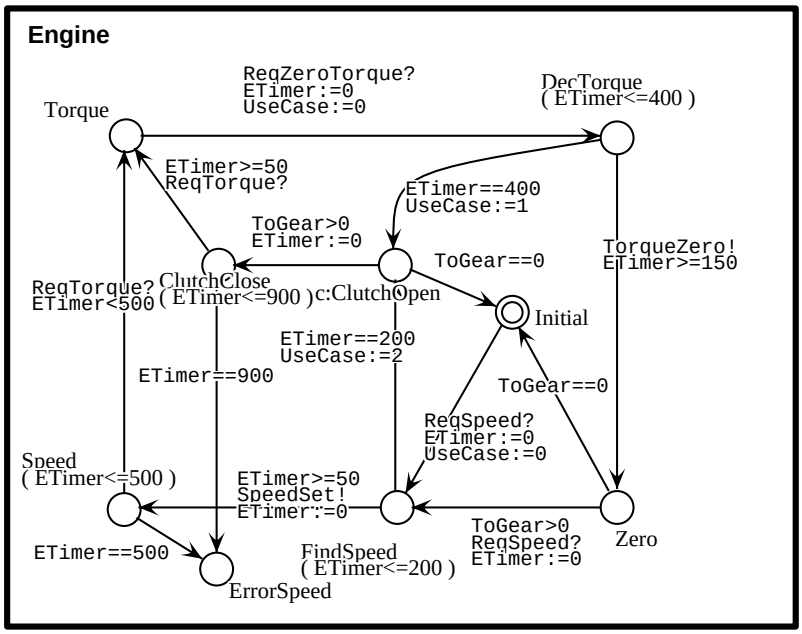


Figure 6: The Engine Automaton.

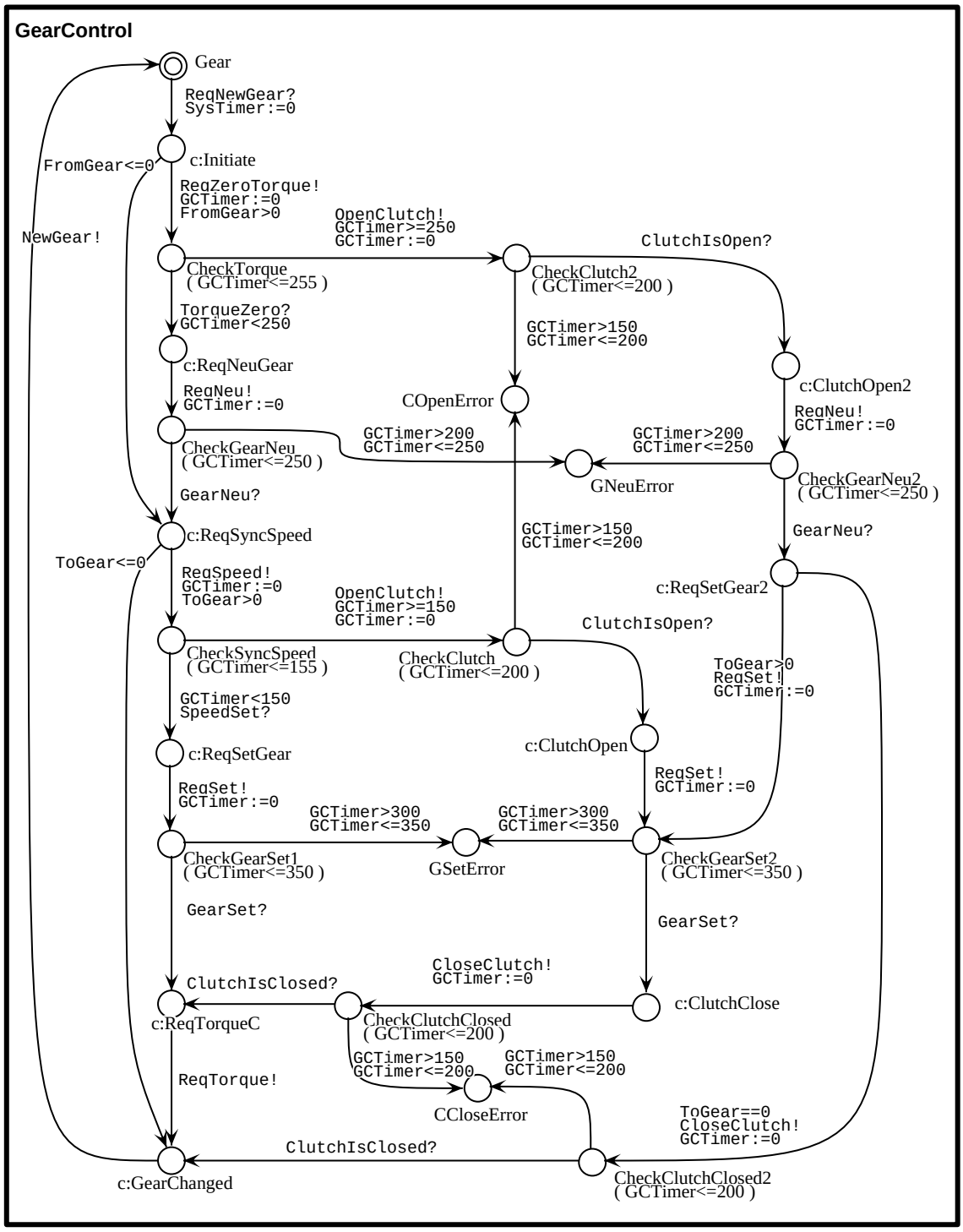


Figure 7: The Gearbox Controller Automaton.

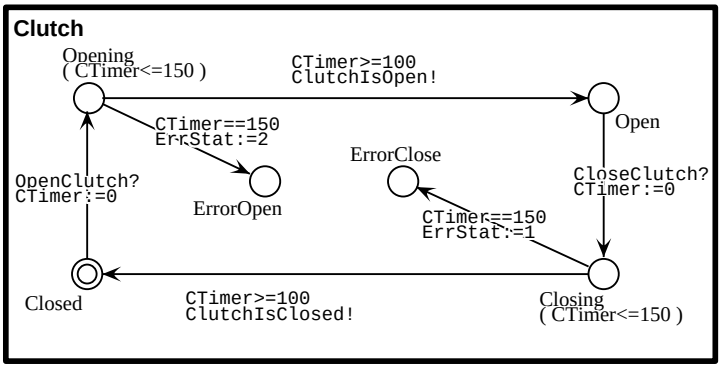


Figure 8: The Clutch Automaton.

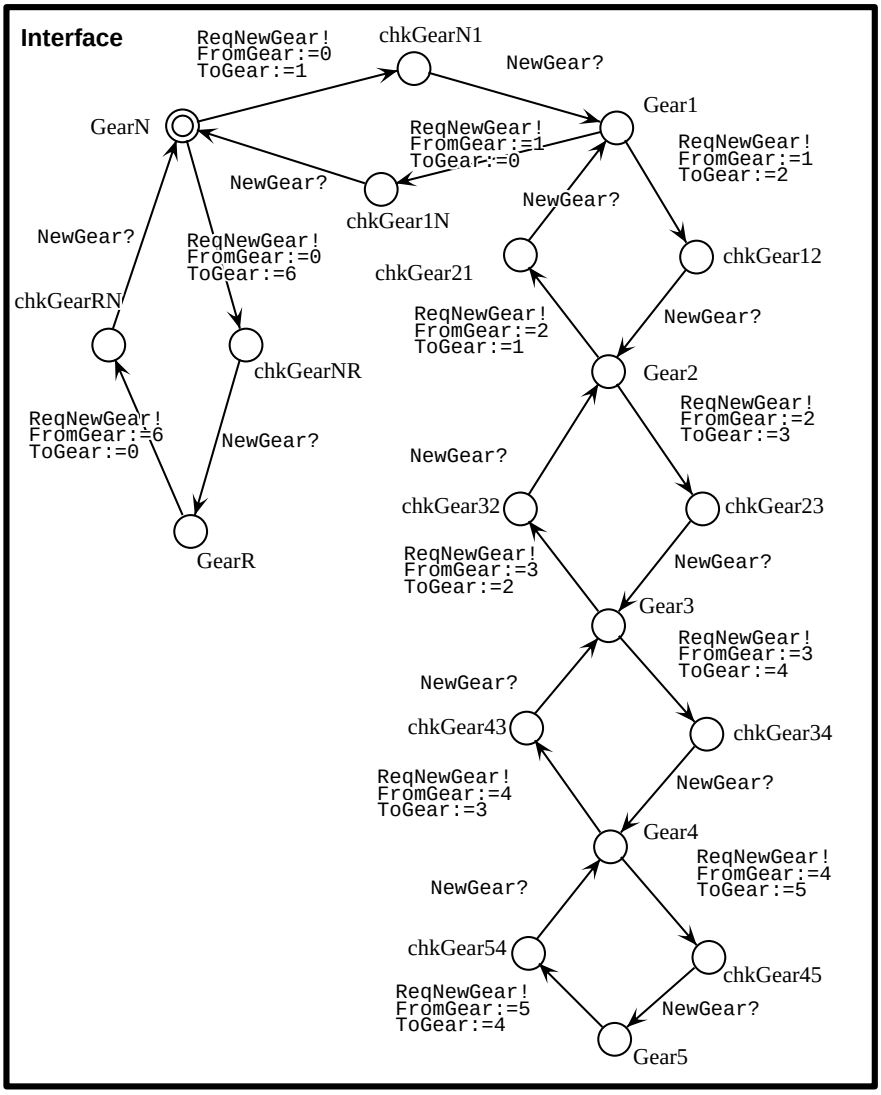


Figure 9: The Interface Automaton.

Department of Computer Systems Dissertation Series

85/03 Joachim Parrow, *Fairness Properties in Process Algebra*

87/09 Bengt Jonsson, *Compositional Verification of Distributed Systems*

90/21 Parosh A. Abdulla, *Decision Problems in Systolic Circuit Verification*

90/22 Ivan Christoff, *Testing Equivalences for Probabilistic Processes*

91/27 Hans A. Hansson, *Time and Probability in Formal Design of Distributed Systems*

91/31 Peter Sjödin, *From LOTOS Specifications to Distributed Implementations*

93/37 Linda Christoff, *Specification and Verification Methods for Probabilistic Processes*

93/40 Mats Björkman, *Architectures for High Performance Communication*

94/46 Fredrik Orava, *On the Formal Analysis of Telecommunication Protocols*

96/70 Lars Björnfot, *Specification and Implementation of Distributed Real-Time Systems for Embedded Applications*

97/80 Bengt Ahlgren, *Improving Computer Communication Performance by Reducing Memory Bandwidth Consumption*

98/98 Björn Victor, *The Fusion Calculus: Expressiveness and Symmetry in Mobile Processes*

98/100 Ernst Nordström, *Markov Decision Problems in ATM Traffic Control*

99/101 Paul Pettersson, *Modelling and Verification of Real-Time Systems Using Timed Automata: Theory and Practice*