

Continuous Modelling of Real Time and Hybrid Systems: From Concepts to Tools

K.G. Larsen¹, B. Steffen², and C. Weise¹

¹ BRICS*, Department of Computer Science, Aalborg University, Denmark

² Lehrstuhl Informatik V, University of Dortmund, Germany

the date of receipt and acceptance should be inserted later

Abstract. The past decade has witnessed a rapid development in the field of formal methods for the specification, analysis and verification of real time systems. Particularly striking is the progress in continuous time modelling, which, despite its unquestioned expressiveness, turned out to be surprisingly tractable: practically relevant classes of continuous time systems can fully automatically be analysed and verified. This has led to the development of a number of corresponding analysis and verification tools of different application profiles. In this paper we concentrate on the two key concepts underlying these tools, known as *Timed Automata* and *Hybrid Systems*. Their role can best be appreciated in the context of formal methods in general, and specifically of specification of real time systems in terms of tailored process calculi and real time logics. All these concepts will be presented in an intuitive fashion, while avoiding as much formalism as possible.

1 Motivation

What this article is about:

- Continuous Time Modelling
- Formal Methods: a key towards Automation
- The Essence of Discrete Time Modelling
- Timed Automata-based Modelling
- The Power of Drifting Clocks
- The Nature of Hybrid Systems
- The Tools KRONOS, UPPAAL, HYTECH

Real time and hybrid systems cross your way several times a day: the automatic teller machine, your car's anti skid system, your video-recorder and washing machine are examples thereof. All these systems share one

* BRICS: Basic Research in Computer Science, Centre of the Danish National Research Foundation

Real Time System: those systems in which the correctness of the system depends not only on the logical result of the computation but also on the time at which the results are produced.

Fig. 1. 'Folk Definition' of Real Time Systems

characteristic: their more or less complicated machinery is controlled by one or more hidden devices. These devices form what computer scientists and electrical engineers call an *embedded system*: it runs as a controlling system within a (complex, typically heterogeneous and reactive) environment (the machinery, the sensors, and the actuators) without being accessible from the outside.

Usually, embedded systems contain at least a highly specialized micro-controller, and often one or even several microprocessors, which are all digital by nature. In turn, the activities in the environment which must be controlled by the processor can be digital or analogue: entering your PIN code into the teller machine is a digital process, while dispensing the money is clearly of analogue nature.

The overall systems thus often form so-called *hybrid systems*, which are characterized by the interplay between discrete and analogue behaviours. *Real time systems*, which can be characterized as stated in Fig. 1 (cf. e.g. [22, 68, 71]), are typical examples of hybrid systems. As timeliness is a central issue in real time systems, it is essential that the timing constraints of the system are guaranteed to be met.

Due to the steadily increasing industrial pressure a lot of research and development effort has been invested on real time and hybrid systems over the past decade. In particular the requirements for safety-critical real time systems led to a rapid development of adequate formal methods for their specification, analysis and verification.

Particularly striking has been the progress in continuous time modelling. Having served as a successful

- easy match with practical application scenarios
- foundational link to natural and engineering sciences
- invariance against changes of time scale
- flexible discretization on-demand in tool environments
- uniform framework for real time and hybrid systems

Fig. 2. Benefits of continuous time modelling

paradigm in physics and engineering sciences for more than 300 years, starting with the discovery of the differential calculus by Leibniz and Newton at the end of the seventeenth century[16], the *continuous* interpretation of time was overwhelmed by the ‘digital revolution’ in telecommunication and least hardware design: continuous effects were discretized through sampling techniques for tractability reasons.

However, it turns out that continuous modelling, despite its unquestioned expressiveness, is in fact surprisingly tractable: a growing number of practical relevant continuous time systems – including system emerging from the area of control theory – can now be analysed and verified fully automatically using classical computer science techniques.

Key to the continuous time analysis and verification tools are the *region* and the *polyhedra technique*, which essentially realize a discretization *on-demand* of the underlying continuous models. Besides enabling the adaptation and transfer of a wealth of discrete analysis and verification methods to the continuous world, this flexible discretization has a strong impact on the tools’ performance: in contrast to the typical situation in discrete time modelling, different components of the system as well as different execution stages can be treated each on its appropriate level of time granularity.

Discretization on-demand also gracefully supports the two central design methods used in system development: *composition* – putting systems together by combining simpler ones – and *refinement* – starting with a rather rough specification and adding details along the development process. In both cases a need to change the (conceptual) time scale may arise. Whereas in the discrete case with its explicit modelling of the time scale this typically requires a complete re-modelling of the system’s components, the on-demand discretization for continuous models automatically takes care of the new situation.

Thus there are very strong reasons for continuous time modelling even of digital systems (Fig. 2).

This paper gives an intuition-guided introduction into the theory underlying KRONOS, UPPAAL, and HYTECH, three outstanding tools for the analysis and verification of continuous time systems, which are also presented in the further articles of this special section. These three tools were designed with different profiles in mind: UPPAAL’s design was mainly efficiency guided, whereas KRO-

borderline of automatic continuous time analysis. In fact, the designers of HYTECH even decided to sacrifice the guaranteed termination of their analysis and verification procedures in order to extend the application scenario: HYTECH is the only system explicitly aiming at the analysis and verification of hybrid systems.

Technically we concentrate on the two key concepts underlying these tools, known as *Timed Automata* and *Hybrid Systems*. The role of these concepts can be best appreciated in the context of formal methods in general and specifically of specification of real time systems in terms of tailored process calculi and real time logics.

All the required concepts will be presented in an intuitive fashion, while avoiding as much formalism as possible. In particular, we will prefer definitions by examples to formal definitions, and refer to the literature for formal details. Thus rather than assuming any specific knowledge about real time systems or computer science, we will try to build upon everyone’s intuition about the nature of real time systems.

Despite this focus, we will also try to provide a more general, but by far not complete view of continuous time modelling. For a comprehensive survey on the theory of real time systems the reader is referred to [11] in the same issue.

We proceed as follows: first we will address the status of *formal methods* in the context of real time system research in Sect. 2, and subsequently give an introduction into *finite state machines* and how they can be extended to include *discrete time* in Sect. 3. Thereafter we discuss continuous approaches to the modelling of real time systems in Sect. 4 and *timed automata*, which use *exact clocks* to measure delays between events, in Sect. 5. Section 6 will then present timed automata with *drifting clocks*; here clocks can have a slight tolerance or uncertainty in their measurement of time, modeling the situation with different clocks in a distributed environment more accurately. Subsequently, Sect. 7 will show how timed automata can be extended to *hybrid systems*, a very general model which allows the description of arbitrary *physical behaviour* as a function over time. The successive Sect. 8 then gives a short overview over the tools KRONOS, UPPAAL and HYTECH which are all presented in detail in articles on their own. Section 10 finally discusses the state-of-the-art of continuous time modelling and future perspectives.

2 Formal Modelling of Real Time Systems

For a long time no methodological progress in the development of real time systems was registered: most designs were ad hoc, spanning from simple products of trial and error on the bad end to well-thought but non-generalizable designs on the good end. The problems of

- Specification and Verification
- Scheduling
- Operating Systems
- Programming Languages and Methodologies
- Distributed Data Bases
- Fault Tolerance
- System Architecture
- Communication

Fig. 3. The Real Time Arena

real time system design seemed to be either scientifically uninteresting, or already solved, or not even suited for scientific methods at all.

John Stanković thought-provoking articles of 1988 about the scientific status of real time systems ([71, 72]) reflect the awareness which had grown in a lot of people at that time. Since then, the situation of real time system design research has changed drastically: many conferences and several journals devote attention to or even focus on this area.

An important step was Stanković’s classification of research subjects in real time system design (Fig. 3). The contributions of this complete special section fall into the category *specification* and *verification*, which is part of the area known as *formal methods* (FM)¹.

The basic motivation behind formal methods is that any rigorous reasoning about systems must be based upon an unambiguous description. This is particularly true if one wants to (partly) automate such a reasoning. Therefore a lot of research has been invested in the development of adequate formal specification formats and languages. Most prominent examples are high-level programming languages, whose development often originally merely aimed at specification purposes, but which were later on supported by compilers.²

The key point of formal description techniques is their mathematical exactness: it is unambiguously clear how the specified system is going to ‘behave’. Exactness should, however, not be confused with precision: “the system must respond within at least 1 and up to 20 seconds” is exact, although one might argue that it is not precise. In an exact specification the amount of imprecision must be explicitly addressed.

Just having a formalism is not very useful in itself. It should come with a methodological support for the better understanding of systems. Wolper [78] distinguishes *weak* and *strong* formal methods. Whereas the latter are characterized by providing *tool* support of analysis and verifications problems, the former simply provide a mathematical framework for formal reasoning. Thus before

¹ The article on Aldébaran([20]) in this issue provides an alternative introduction into formal methods.

² Standard ML [1] is a typical language with such a history.

qualifying questions:

- What is the formalism?
- What are the methods?
- How far can the methods be automated?

In the remainder we will consider these questions for strong formal methods which have been developed for continuous real time systems. The other articles of this special section will then explain how these methods can be automated and how they are realized in KRONOS, UPPAAL, and HYTECH.

3 The discrete approach

- Formal Methods
- Finite State Machines
- Reactive Systems
- Temporal Logics
- Model Checking
- Qualitative and Quantitative Time

In order to understand the complex timed formalisms, it is best to start with the discrete or “untimed” ones. Many formalisms – in fact all those we will look at in detail – can be translated into *finite state machines* (short: FSMs), a fundamental concept of computer science and electrical engineering. An FSM has a *finite set of states* – hence the name – and a description how the machine changes from one state into another. Switching state is called a *transition*. An imported feature of FSMs is that they can easily be visualized: draw the states as nodes of a graph (i.e. circles) and the transitions as directed edges between them (i.e. arrows), and label them appropriately. A *flowchart* is a well-known representation of the control part of a program as an FSM.

In Fig. 4 we give a specification of a classical book example from the formal methods world: the coffee-or-tea machine (see e.g. [54]). Our version will output a tea or a coffee, depending on the user’s choice, after pressing the appropriate button.

The behaviour of this machine is as follows: in the idle state, the machine waits for the user to push either the coffee or the tea button. After the user has made his/her choice, the machine will first deliver a cup. Then the machine will fill the cup with coffee or tea – depending on the user’s previous choice – and return to its start state.

The overall behaviour of this machine can be described by its legal *traces*: traces are just the inscriptions of all maximal paths beginning with the start state of the machine. So a legal trace of the machine will either start with an inscription

push coffee button.deliver cup.deliver coffee

or

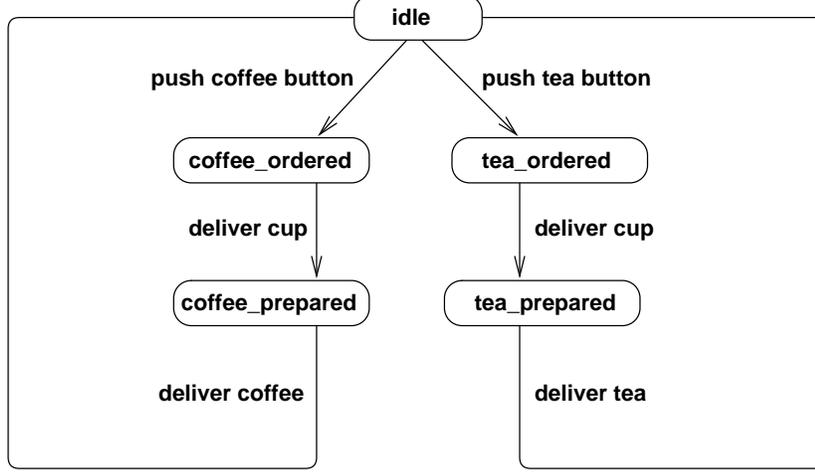


Fig. 4. The inevitable coffee machine

push tea button.deliver cup.deliver tea

In fact, any legal trace will be an infinite sequence³ of an arbitrary mixture of these two inscriptions.

For FSMs, there is a large and established theory for specification, analysis and verification, which dates back to the fifties. Prominent examples are the theory of finite automata and formal languages, which is extremely successful e.g. in compiler construction.

The “untimed” theory has been applied in practice, e.g. for hardware design as well as in the design of communication protocols and digital controllers, and it is becoming more and more widely spread. A very important concept in this theory is the idea of *reactive systems* (see [51] for a good tutorial). A reactive system is characterized by its communication with some environment. The system reacts to input from the environment by supplying corresponding output.

The coffee machine is a reactive system which responds to inputs “push coffee/tea button” and has outputs “deliver cup” and “deliver coffee/tea”. These inputs and outputs are often called *actions*.

A standard way to speak formally about systems like the coffee machine is to describe its behaviour by the set of all legal traces. In the following we will subscribe to this trace point of view, which in fact reflects the user’s viewpoint most directly. Alternative approaches, like the ones based on *bisimulations* [52], *failures* [44] or *testing* [55] are used to describe important properties of concurrent systems, e.g. deadlock potential, which are not expressible using traces.

A key question in formal methods (and in practice as well) is: does the implementation meet its specification, i.e. is it a proper implementation? In our setting this question can be dealt with in the following way:

³ assuming the machine will exist forever

first model both the behaviour of specification and implementation as FSMs, and then check if every trace of the implementation is also a trace of the specification, i.e. test whether the traces of the implementation are allowed by the specification. Mathematically, this means that the set of traces of the implementation must be included in the set of traces of the specification. Verification then becomes checking *trace set inclusion*.

Trace set inclusion can be checked automatically for FSMs. However, the approach suffers from its high *worst-case complexity*, the time spent on verifying trace inclusion automatically may grow exponentially in the number of states of the systems. In fact there are systems of comparatively small size, which cannot be automatically checked for trace inclusion. However, luckily, most of the practical systems escape the exponential worst-case complexity, making automatic trace inclusion checking a tool of practical relevance.

FSMs provide a formal yet intuitive way for describing (reactive) systems. In particular, they provide a formal basis for the *analysis* of reactive system, which is concerned with answering vital questions about the systems’ behaviour like “will I get coffee, after pushing the coffee button?” Answering such a question means to test if the system has a certain *property*. In a formal method, *logics* are usually used to describe such properties in terms of *formulae*.

If a system has a certain property, it is then said that the system *satisfies* the corresponding formula. Logicians also say that the system is a *model* for the formula. Therefore the task of testing if a system is a model for a formula is called *model checking*. Model checking is one of the most successful methods for formal system analysis.

In the untimed case, there is a large family of logics suitable for the analysis of reactive systems which

in these logics as questions about correct sequencing in traces (e.g. will I always get a coffee *after* I inserted enough coins?). Many of these logics can be analysed (semi-)automatically, and there exist a lot of powerful tools for this purpose.

In the case of real time systems, we are not only interested in the correct (temporal) order of events, but additionally in the (explicit) amount of time passage. So for real time systems, we have to talk about time *quantitatively* instead of just *qualitatively*. For digital systems, where events are driven by some pulse, it is very easy to model the passage of time: just add a special new action “tick” to the language, which models the ticking of the system’s clock. Figure 5 shows how to model the coffee machine in this way. Assuming that a tick lasts 5 seconds, it takes 25 seconds to prepare the coffee and 10 seconds to fill the cup, while it takes 30 seconds to prepare the tea and 15 seconds to pour the tea into the cup.

RTL by Jahanian and Mok (see [53, 45]), TTM/RTTL by Ostroff (see [62, 63]) or the Timed Systems of Hennessy and Regan (see [35]) are prominent examples for the numerous successful discrete time approaches to the modelling and verification of real time systems.

The good point about the discrete approaches is that, in principle, there is nothing new: the models are essentially still FSMs, with just a specially treated action “tick”. Therefore, the known analysis techniques can be applied, and existing tools for reactive systems can be used.

However, as already emphasized in the introduction, there are a lot of good reasons to prefer continuous over discrete modelling. One apparent reason is that it is much more general than the discrete approach. In the following sections, we will give an introduction into continuous approaches.

4 Early continuous approaches

- Process calculi CCS and CSP
- Timed CSP, Timed CCS and ATP
- Instantaneous Actions and Two-Phase Systems
- Real Time Logics

The first approaches that marry discrete and continuous behaviour can be found in the world of *process calculi*. These process calculi are a way to describe distributed reactive systems algebraically. Therefore they are often referred to as *process algebras*. They have been very successful in the formal methods world because they allow an easy and concise mathematical treatment while being very intuitive at the same time. Milner’s CCS (Calculus of Communicating Systems, [52]) and Hoare’s CSP (Communicating Sequential Processes, [44]) are among the foremost examples of process calculi. Dialects of these

are tools supporting specification and analysis in CCS or CSP like the Concurrency Workbench⁴[26] and FDR⁵[67], respectively.

In process calculi, equivalences between processes and preorders on the processes play an important role. Equivalences allow one to compare two different processes to find out if they behave in the same way, while preorders allow for determining if one process is an implementation of the other due to the fact that it satisfies all the requirements posed by the specifying process.

A very early example is *Timed CSP* – a timed version of CSP – by Reed and Roscoe (see [66]). Later came Wang Yi’s *Timed CCS*, a real time extension of CCS with a continuous time domain [79, 81, 80]. Another prominent example of a real time calculus is ATP (Algebra of Timed Processes, [59]), which has a discrete and a continuous interpretation.

All these approaches have something in common: the “normal” actions are treated as being instantaneous (i.e. taking “no time”), while the new ingredient is an operator to express delays of some duration. Although such an approach might not match the intuition of most people at first sight (everything seems to take *some* time), it turned out that this idea is a good theoretical concept, as Nicollin and Sifakis convincingly argue in [58].

One reason to use an approach like this is its generality: activities which take some time to happen can be modelled by a distinct start and stop action (which are both instantaneous) and the appropriate delay in between. Moreover, the abstraction leading to a modelling with instantaneous actions and delays in between has already successfully been used in other disciplines. In electrical engineering, for example, up- and down-going edges of pulse diagrams, which describe the behaviour of wires in a digital system, are often assumed to be instantaneous, although raising or lowering the voltage clearly takes some time.

The distinction between instantaneous actions and delays, which model the passage of time, leads to a *two phase* behaviour: phases during which a single or more actions occur together with state changes while no time passes are interleaved with phases where time passes while nothing else happens, as illustrated in Fig. 6.

Process calculi equipped with some time domain allow to specify real time systems, and one can even reason about the systems using the methods which were already known within these calculi. Independently, at about the same time, several real time logics (see [3, 33, 8, 9] and [10] for a survey) were developed which followed similar ideas. These logics are tailored for the formulation of vital properties of real time systems, but they may also be used to specify the overall behaviour of such systems.

⁴ <http://www.dcs.ed.ac.uk/home/cwb/>

⁵ <http://www.formal.demon.co.uk/FDR2.html>

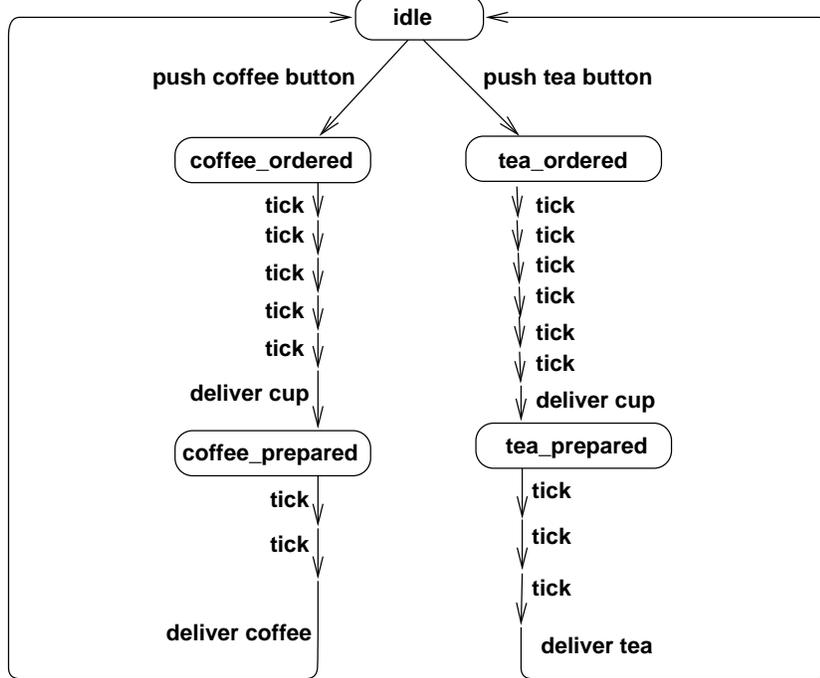


Fig. 5. A discrete time coffee machine

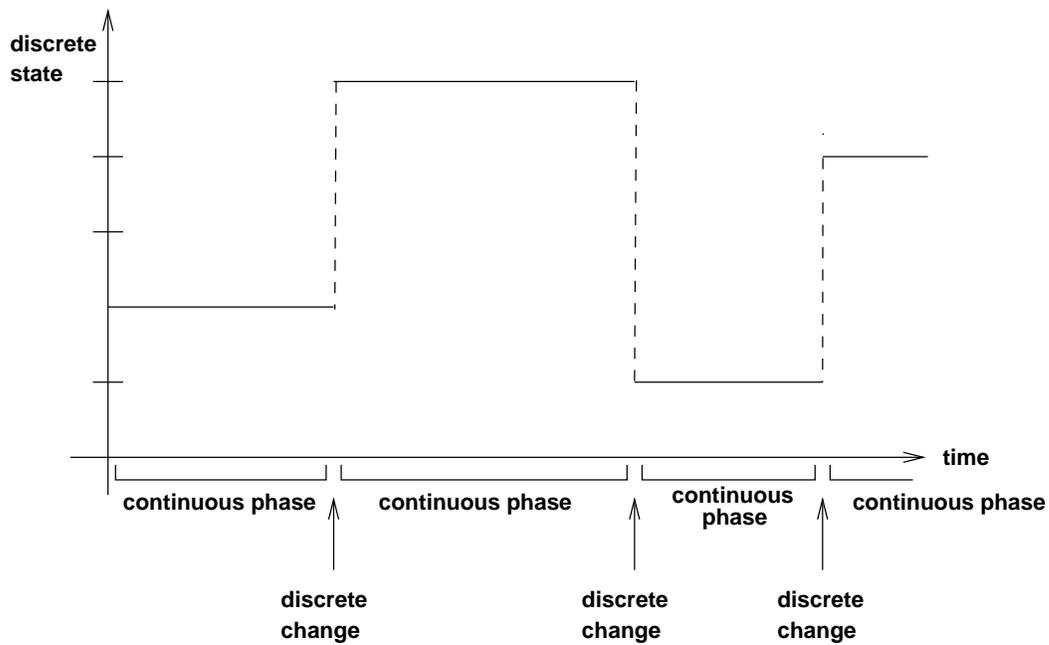


Fig. 6. Two Phase System

The practical relevance of these real time calculi and logics depends strongly on how far reasoning in them could be automated. This raises the question of *decidability* of the equivalences, preorders and logics, the necessary condition for the existence of adequate corresponding tools.

The next section will introduce the notion of timed automata and explain the principles behind the region

technique which provides the formal foundation for answering these questions.

- Timed Automata
- Region Technique
- Reachability Algorithm
- Capabilities and Limits of Timed Automata

This section introduces *timed automata*. The main idea behind this formalism is the introduction of a finite set of clocks for measuring delays and controlling the execution of actions which have guards that depend on timing information. In this section, all clocks are meant to measure time *exactly*, i.e. they all run at the same speed as some fictitious global clock measuring the “real time”.

5.1 The Basics

The theory of *timed automata* was introduced in the thesis of Rajeev Alur ([2]) and related publications ([3, 4]). They provide a useful FSM model for the specification of real time systems.

Figure 7 displays the timed automaton for a continuous time version of our coffee machine, whose associated set of clock consists just of one clock X . The transitions of this automaton are labelled with three distinct items: a *guard*, an *action* and a *reset assignment*. As an example, the “deliver cup” action now consists of the guard $X=30$, the action deliver cup itself, and the reset assignment $X:=0$.

The start state of the machine is called *idle*. In this state the machine is waiting for an arbitrary time for the user to push a button, so the guard of the “push tea/coffee button” actions is just *true*. After the coffee button is pushed, the machine enters the state *coffee_ordered* and the clock X is set to zero. The machine needs 25 seconds to boil the water in this state. When the water is boiling, the machine will deliver the cup. So the guard for the “deliver cup” action is $X=25$. After the 25 seconds are expired, the machine will go from *coffee_ordered* to *coffee_prepared*: the coffee is now ready to be delivered. It takes 10 seconds to fill the cup with coffee. Thus the guard of the “deliver coffee/tea” action is $X=10$. After delivering the coffee the machine goes back to its start state *idle*. Now the user can push e.g. the tea button, and tea will be delivered analogously, except that the guards are slightly different.

Formally, a timed automaton is an FSM with an associated finite set of clocks and specially labelled transitions. A transition label consists of an action, a guard and a reset assignment. The reset assignment is just an assignment of zero to a subset of the clocks. We often call this subset of the clocks the *reset set*. The guard is a comparison of a clock against an upper or a lower bound; such comparisons can be combined by conjunction. The bounds can be strict or non-strict, and they should be rational numbers. See Fig. 8 for an overview.

Timed Automaton =
Finite State Machine + Finite Set of Clocks

Nodes are called *Locations*

Transitions are labelled with

- action (what is done?)
- guard (when is it done?)
- reset-set (which clocks are reset?)

Fig. 8. What defines a Timed Automaton?

Timed automata can be executed as follows: in the beginning all clocks are set to zero, and from then on they all increase uniformly as time progresses. A transition can only be taken if its guard is fulfilled. When taking a transition, all clocks in the reset set are set to zero, while the rest retain their values. This intuition behind the execution of a timed automata can be made precise by giving the automaton a *semantics* in terms of a labelled transition system. The particular version used here is called a *valuation graph*. States (nodes) of a valuation graph consist of a location of the timed automaton, and a valuation of the clocks, i.e. an assignment of real values to clocks. The transitions of the valuation graph are labelled in the “ordinary” way either by an action or by a delay (which is a positive real number). Figure 9 gives a path in the valuation graph of the timed automaton from Fig. 7. Note that we write clock valuations as formulae $X = a$. Thus a state of the valuation graph has the form *node*, $X = a$.

While, in principle, we allow arbitrary rational values in the bounds in the timed automaton, we will assume that all bounds are natural numbers for the rest of the paper in order to simplify the representation. This does not pose any restriction, as timed automata are finite, and therefore only possess a finite number of bounds. For a finite set of rational numbers, however, it is always possible to find a factor transforming all these numbers to integers by multiplication. This multiplication only changes the time scale, without affecting the behavioural properties of the automaton.

There are two main results given in Alur’s thesis ([2]): first, trace inclusion as a relation between specification and implementation is discussed for timed automata, and it is shown that trace inclusion is undecidable for timed automata, contrary to the untimed case. Second, it is shown that model checking for the real time logic TCTL is still decidable. So while on the one hand – maybe unsurprisingly so – trace inclusion checking, a standard verification technique (cf. Sect. 3), cannot be automated for timed automata, on the other hand, model checking can be fully automated. This decidability result opened the door for the computer assisted analysis of continuously modelled systems. The corresponding model checking algorithm is based on the *region technique* described in the

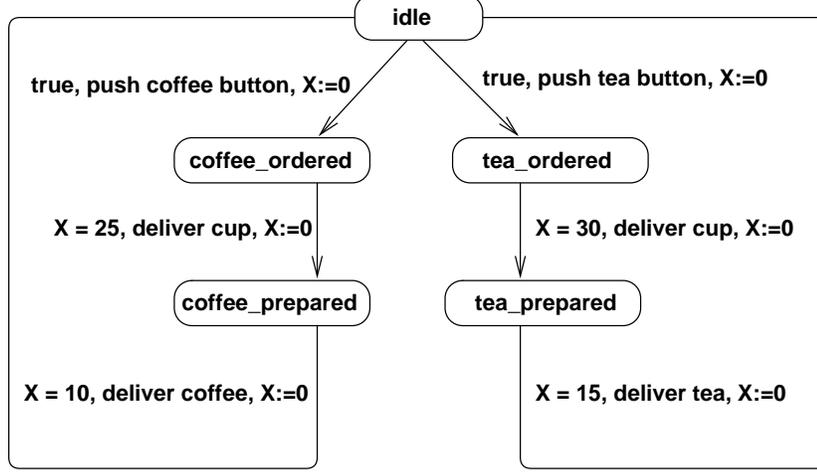


Fig. 7. A continuous time coffee machine

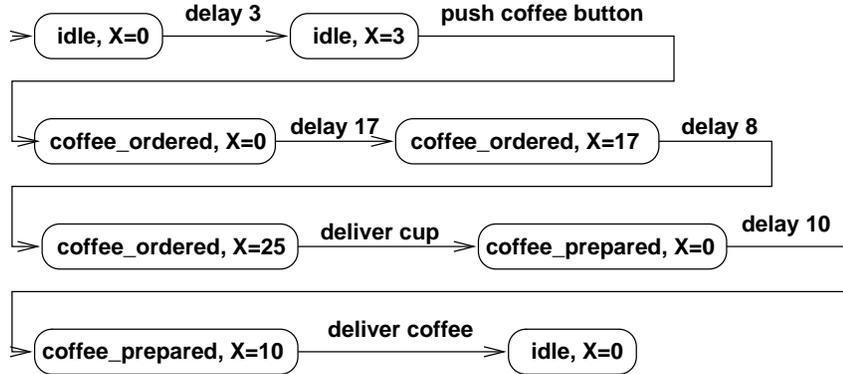


Fig. 9. Path of the continuous time coffee machine

next subsection, which can be regarded as the key to the success of timed automata.

5.2 The Region Technique

The problem with timed automata is that any reasonable analysis is in principle based on the corresponding valuation graph, which is in general infinite (if you are familiar with graph theory: it is even infinitely branching). Luckily, instead of using the valuation graph, it is sufficient to use the *region graph* of the timed automaton.

The main idea of the region technique is that it is possible to find a finite representation of the valuation graph which represents all the necessary reachability information symbolically. This finite representation is called the *region graph* and can be computed effectively. In the rest of this subsection, we will explain the region technique in more detail. As things become rather technical, one may consider to skip this subsection on first reading.

We start the explanation of the region technique with a simple example. The timed automaton given in Fig. 10 has one clock X , which controls the time at which an action a can happen. The action is allowed to occur within

an interval of one (inclusively) to two (exclusively) time units after the process started, as required by the guard $1 \leq X < 2$. After a is taken, the clock will be reset to zero again, and no other action is possible then.

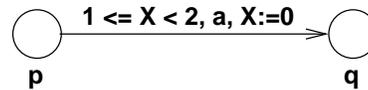


Fig. 10. Sample Timed Automaton

Figure 11 gives the initial part of the region graph of the process concentrating on the potential of state p . The idea here is the following: if we restrict ourselves to natural numbers in the guards, then for the clock X it is only important to know if X has a certain natural value, or if X is somewhere between a natural value n and its successor $n + 1$. So instead of looking at all possible values for X , we only need to distinguish the intervals $[0, 0]$, $(0, 1)$, $[1, 1]$, $(1, 2)$, and so on. Alternatively these intervals can be described as logical formulae of the same kind as the guards: $X = 0$, $0 < X < 1$, $X = 1$, $0 < X < 2$

are in fact the regions we are interested in.

The nodes of the region graph are pairs of a location and a region. Figure 11 shows the region graph for the automaton from Fig. 10. The node $(p, 0 < X < 1)$ is an example of a pair of a location p and a region $0 < X < 1$. Due to the two-phase nature of the valuation graph, there are two ways to go from one node to the other: either by letting time pass (e.g. $(p, 0 < X < 1) \xrightarrow{\text{time}} (p, X = 1)$) or by some action (e.g. $(p, 1 < X < 2) \xrightarrow{a} (q, X = 0)$).

The idea of using intervals of the form $X = n$ and $n < X < n + 1$ (where n is some natural number) would still lead to infinite graphs. Additionally, we take into account that the timed automaton itself is finite, therefore there must be a maximal number k to which clocks are compared. If the clock exceeds this value k , then we are not interested in the exact value anymore, as any guard will either be fulfilled for all or for none of the values $X > k$. Thus it is sufficient to use intervals $X = n$ and $n < X < n + 1$ for $n < k$ and additionally the intervals $X = k$ and $X > k$.

In the case of our simple example, the value for k is 2. So the set of all regions is $\{X = 0, 0 < X < 1, X = 1, 1 < X < 2, X = 2, X > 2\}$, which is clearly a finite set.

In the case of more than one clock, things are not that easy. However David Dill noted that it is sufficient to keep track of the values of all clocks and of the differences of all pairs of clocks. In [31] he introduced *difference bound matrices* (short: dbm) which can be used to store this information for a region.

Instead of using dbm's, one can also represent regions by formulae of a very specialized kind. Let again be k the maximal constant appearing in the formulae of a given timed automaton. A formula for a region consists of conjunctions of comparisons. All admissible comparisons can be found in Fig. 12. There is a component of the formula for each clock C and for each pair of clocks C, C' (where $C \neq C'$). A single clock is either above k , or it is equal to a natural number below k , or it is within an open interval of two natural neighbours n and $n + 1$ both less than k . The same is true for the clock differences, but additionally we allow differences to be negative (i.e. $C - C' < 0$). Clock differences are only needed when both clocks are below k . Figure 13 lists all regions if the clock set is $\{X, Y\}$ and $k = 1$.

There is a subtle problem with the representation of regions as dbm's or as specialized formulae: if we want to do computations on regions, then the "canonical form" as given above might be destroyed. A major result of Dill in [31] was how to (re-)compute the canonical form⁶. The canonical form is very important in order to compare regions, as in principle, the same region can have different representations.

⁶ Although later it was noted that Dill's method had already been proposed in [17] in a different context.

- $C = n$, or
- $n < C < n + 1$, or
- $C > k$

for each pair of clocks C, C' a term of the form

- $C - C' < 0$, or
- $C - C' = n$, or
- $n < C - C' < n + 1$, or
- $C - C' > k$

Fig. 12. canonical form for regions

$$\begin{aligned}
 & X = 0 \wedge Y = 0 \wedge X - Y = 0 \\
 & 0 < X < 1 \wedge 0 < Y < 1 \wedge X - Y = 0 \\
 & X = 1 \wedge Y = 1 \wedge X - Y = 0 \\
 & X > 1 \wedge Y > 1 \\
 \\
 & X = 0 \wedge 0 < Y < 1 \wedge 0 < Y - X < 1 \\
 & 0 < X < 1 \wedge 0 < Y < 1 \wedge 0 < Y - X < 1 \\
 & 0 < X < 1 \wedge Y = 1 \wedge 0 < Y - X < 1 \\
 & X = 1 \wedge Y > 1 \\
 \\
 & X = 0 \wedge Y = 1 \wedge Y - X = 1 \\
 & 0 < X < 1 \wedge Y > 1 \\
 \\
 & X = 0 \wedge Y > 1 \\
 & \text{(and symmetrically with } X \text{ and } Y \text{ exchanged)}
 \end{aligned}$$

Fig. 13. All regions for $\{X, Y\}$ and $k = 1$

Many algorithmic problems for timed automata can be solved using the representation of the valuation graph by a region graph. The *region technique* is one of the key ingredients for many analysis algorithms for timed automata.

5.3 Extensions of Timed Automata

Several extensions of timed automata have been proposed. We mention two important ones.

Xavier Nicollin showed in his thesis ([56]) that in the guards comparisons of *clock differences* against rational numbers can be permitted without a big change in the region technique. It is worth noting that he also showed how to translate the timed process calculus ATP into timed automata (see [60]).

Figure 14 gives a simple example of a timed automata containing a guard with a clock difference. Note that the b action can only be taken if the a transition was taken at least once after more than two time units after process start. If the a action has not been taken at all, then X and Y are always the same, so the guard $X - Y > 2$ cannot be true. If the a action is taken at some time t , then afterwards $X - Y = t$, so t must exceed 2 to fulfil the guard.

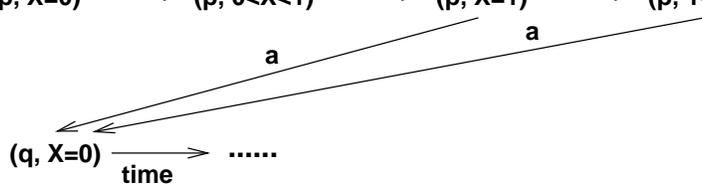


Fig. 11. Region Graph of Sample Timed Automaton

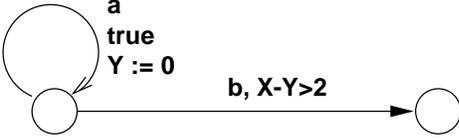


Fig. 14. Example of a timed automaton with clock differences

Henzinger et al. introduced the idea of invariants into timed automata (in [41]) and called this variant *safety timed automaton*. An invariant is a formula of the same type as the guards. While the guard is associated with a transition and controls when the transition can be taken, an invariant belongs to a location and controls how long the system is allowed to stay within the location. Figure 15 shows an example of a safety timed graph. The automaton can only stay within the left location as long as clock X is less than or equal to 4. So the b action, which can in principle be taken all the time, must be taken when X reaches 4, while the a action, which can also be taken all the time, need not be taken at all. We shall see later that invariants also occur in the generalization of timed automata.

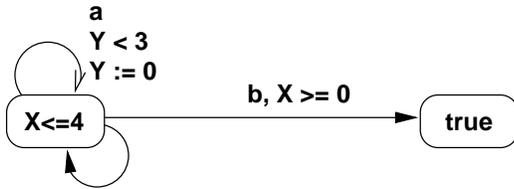


Fig. 15. Example of a safety timed automaton

5.4 Main Results

The main result of Dill, Alur et al. was that automatic analysis of real time systems was still possible even when time was modelled continuously ([2, 3, 4, 7]). The necessary ingredient to reach this means is the observation that the notion of region induces a *finite* partitioning of the state space, thus leading to a finite, abstract model which is still fine-grained enough to maintain all the reachability information.

So how is this finite model used in the algorithms? In principle, all algorithms are based on a *reachability anal-*

ysis of the valuation graph. Model checking e.g. can be done in terms of reachability by asking “will the system ever reach a state where the formula is violated?” So in general, all that needs to be done by the tools is an enumeration of all reachable states. Note that in the case of timed automata, this cannot be done directly from the timed automaton itself. Using the valuation graph on the other hand, reachability questions become trivial: a state is reachable, if there is path from the start state to the state itself within the valuation graph. But as the valuation graph is infinite (in fact, very infinite), we cannot construct it directly with a computer.

The next paragraphs get a little technical again, as we will explain a general algorithm to find all the reachable states of a timed automaton. This algorithm is basically a graph exploration algorithm with some minor (but important) ingredients from the theory of timed automata. The main idea is to go through the graph step by step: so in the first place one is interested in what is reachable from a given state within one step. A step in a timed automaton can be either a time-step or an action a . This distinction (which is just the two-phase character of valuation graphs) will be used in the algorithm.

Let us start from a fixed node of the region graph. Let us say that this node is (p, γ) , where p is the current location and γ is the current region. As γ represents a whole set of clock valuations, so (p, γ) represents a whole set of states of the valuation graph. On regions, there is some successor relation defined which describes the next region which is reached by letting time progress. So by time steps we will reach a node (p, γ') , where γ' is the time successor of γ . Note that γ' is a region as well. We call (p, γ') the *time successor* of (p, γ) and will write $\text{succ}_\chi(p, \gamma)$ for it⁷. Note that sometimes $\text{succ}_\chi(p, \gamma)$ can be empty.

In the example in Fig. 11, the successor of $(p, 0 < X < 1)$ is $(p, X = 1)$, as $X = 1$ is the next region reached by letting time pass, i.e. by increasing clock X . The state $(p, 1 < X < 2)$ has no time successor, as the process must leave p using the a transition. Thus time is not allowed to progress further than until some point in $0 < X < 2$, and therefore the time successor is empty.

Now for every transition e (for edge) leaving the location p in the timed automaton, there is a node (p', γ')

⁷ The idea of using χ to indicate time is from [60] and stands for Kronos, the Greek god of time.

transition e , and γ' is computed from γ by resetting the appropriate clocks. Note that γ' is only non-empty if the guard of e is enabled, and that γ' is again a region. We write $\text{succ}_e(p, \gamma)$ for the node reached from (p, γ) using the transition e .

In the example of Fig. 11, the a -successor of $(p, X = 1)$ is $(q, X = 0)$ as the a action is allowed for $X = 1$ and X is reset along this transition. The node $(p, 0 < X < 1)$ has no a -successor as no a action is enabled when $X < 1$.

Using this notation, we can outline a reachability algorithm for timed automaton as in Fig. 16. The algorithm computes a set \mathcal{R} of nodes (p, γ) , where p is a location of the timed automaton and γ is a region. A state is reachable if it belongs to one of the nodes in \mathcal{R} . The idea is to start with the start node in \mathcal{R} . Then step by step the time successors and the transitions successors of the nodes already in \mathcal{R} are computed, and added to \mathcal{R} if not already there. Nodes in \mathcal{R} for which we have already computed the successors are marked as visited.

- initialize \mathcal{R} with $\{(p_0, \gamma_0)\}$
 - mark all nodes as not visited
 - for all (p, γ) in \mathcal{R} not already visited:
 - mark (p, γ) as visited
 - add $\text{succ}_\chi(p, \gamma)$ to \mathcal{R}
 - for all transitions e leaving p
 - add $\text{succ}_e(p, \gamma)$ to \mathcal{R}
 - until all nodes in \mathcal{R} are marked visited

Fig. 16. Reachability Analysis

The algorithm terminates as soon as there are no more nodes in \mathcal{R} which have to be examined, i.e. if all nodes in \mathcal{R} are marked visited. As there is only a finite number of nodes in a region graph, the algorithm must terminate after a finite number of steps.

So there are three important facts for this algorithm to work properly: First, the result of succ_χ and succ_e , when applied to a region, must again be a region. Second, an efficient way of representing and manipulating is needed (cf. [23]). Third, there may only be a finite number of regions to ensure termination of the algorithm

Thus the region techniques leaves us with a basis for analysis algorithms for timed automata. Therefore continuous time models can be analysed in a way similar to the discrete and untimed case.

However, there is a certain price to pay: while the principal idea of using clocks to express time distance between events is a very natural way of specification, the usage of the clocks in the formalism is quite limited: all clocks are assumed to be exact and the only operation on them is a reset. Further the allowed tests on clocks are very restrictive: only a clock or a difference between clocks can be compared against a natural num-

we shall see some of them in the coming sections) where this is not sufficient to model a system. On the other hand there is a large number of case studies which prove that the model is already useful for real problems.

The central problem in applying the region technique to real problems is the size of the region graph: the number of regions grows exponentially in the number of clocks and the size of the maximal constant used in the automaton. Thus while the region technique solves the problem of analysing timed automata theoretically, it does not yield an efficient solution. Performance is of major importance in tool design, and as we will see later, all three real time tools of this issue of STTT cope with this problem efficiently.

6 Realistic Clocks

- Realistic Distributed Clocks
 - Drift and Synchronization
 - Philipps Audio Control Protocol: an Industrial Example

This section introduces timed automata where clocks are more realistic than the exact clocks of the former section as they are admitted to be inexact within a given tolerance, i.e. clocks have a slight *drift*.

6.1 Timed Automata with Drifting Clocks

Speaking of real problems, there is even a problem with exact clocks: in reality, different clocks in a distributed environment will not run at exactly the same speed, but they will all drift slightly. The longer the system runs, the bigger the absolute error becomes when clocks drift. Thus it seems quite reasonable to take this drift into account when modelling the system. This leads to the idea of *drifting clock timed automata*, which is quite simple: with every clock we associate a *drift*, i.e. a closed interval of the reals. At any time, the speed of the clock must be within this interval. So now we are not speaking of exact clocks any more, which all run at the same speed of 1, but our clocks may vary in their speed within given bounds. A realistic situation e.g. is a clock with a tolerance of 5%. This would be modelled by a clock with a drift of $[0.95, 1.05]$. So at any time the clock may run with a speed of 1, but it may also be up to 5% slower or faster. The model allows individual drifts for different clocks, and the drift can even change for an individual clock at different control locations.

This extended model is not only more realistic, but luckily it is still possible to analyse the model automatically. Sifakis et al. showed in their paper [61] that there is a translation from a drifting clock timed automaton

the same reachability properties. Thus by combining the translation with the region techniques presented in the previous section, one obtains an algorithm for the analysis of drifting clock timed automata.

As the drift of a clock can be an arbitrary interval with rational borders, this also allows the modelling of clocks running at different speeds by choosing drifts like $[2, 2]$ or $[5, 5]$.

In many discussions on modelling real time, not only using exact clocks is attacked as being unrealistic, but also using exact time points in guards like $X = 4$. Note that this can be overcome by using guards of the form $4 - \epsilon \leq X \leq 4 + \delta$, but that in many cases it is already sufficient to model the uncertainty with a small drift of the clock instead of changing the guard.

6.2 Drift and Synchronization

We further would like to point out that no real system will be able to have drifting clocks which are never synchronized. If synchronization would never happen, the absolute error could grow above any measure, and this would mean that the system would break at some time as the measured time is too far away from the really elapsed time. We all know that even our most exact clocks here on earth must be synchronized from time to time. Now synchronization means “to take away the drift”. If you think of a system with exact, discrete clocks, you can think of this system as being synchronized with every clock tick. A drifting clock system is much more realistic, by allowing the clocks to drift between synchronization. Now and then, synchronization will happen when exchanging messages between system components. As already said, a realistic system without this synchronization will break eventually. In this light, the case of exact clock timed automata can be seen as the most extreme case of synchronization: in the case of discrete clocks, there might be a small drift, which is however not observable as we look at the system only when the synchronizing clock tick occurs, while in the case of exact continuous clocks all components are in lockstep at all times.

6.3 The Audio Control Protocol

The *Audio Control Protocol* is a well known case study which shows the usefulness of drifting clock timed automata in practice. In 1994, Frits Vaandrager et al. proved that for a communication protocol for Philips’ audio devices, the protocol was reliable up to a clock tolerance of $1/17$ (see [21]). For Philips it was very important that the protocol was reliable up to a tolerance of 5% (i.e. $1/20$): In order to keep the price of their audio devices low they had to rely on clocks in the devices which were no more accurate than this. Vaandrager was not only able

$1/20$ tolerance but also demonstrated an exact bound on the tolerance of $1/17$. His original proof used a variant of drifting clock timed automata and was done by hand. In the presentation of his paper, Vaandrager said that it would be quite challenging to see if it was possible to find an automated proof.

To his own surprise, already in the next year UPPAAL and HYTECH were able to give automated proofs (see [46, 43]). While the original proof took several months, the automated proofs needed just a few weeks to model the audio control protocol in their input language (see Fig. 17 for an example of the modelling for UPPAAL) and only some hours to verify that the tolerance $1/20$ was sufficient. The first automatic proof took seven hours, but UPPAAL and later also HYTECH managed to do the same later on in eight seconds. HYTECH was even able to prove the upper bound of $1/17$ by parametric reasoning (see [43, 13]).

Vaandrager’s original proof however was based on a simplification of the protocol as it assumed only one sender and receiver so that no collision would occur on the bus. David Griffioen, a student of Vaandrager, gave a hand proof for the case of bus collision in his master’s thesis ([32]). The UPPAAL team together with Griffioen were able to give an automated proof for this as well (see [18]), with a total verification time of no more than eleven minutes.

The audio control protocol marks the first example of the usefulness of these approaches and the tools for real life examples of real time problems.

7 More than Clocks

- General Hybrid Systems
- Continuous Variables
- Boiling Water
- Linear Hybrid Systems
- Polyhedra Technique

Hybrid Systems ([49, 5]) are a formalism for modelling mixtures of discrete and continuous behaviour. They are a very broad generalization of timed automata, where continuous variables are used for modelling arbitrary continuous behaviour instead of just representing clocks. This section will explain the notion of hybrid systems.

7.1 General Hybrid Systems

As we already emphasized in the section on real time systems, a real time system can typically be divided into a *controlling system* (often called the system) and a *controlled system* (the *environment*) [73]. The behaviour of the controlled system is given a priori, although we are

about the environment, and it will require the environment to behave this way, i.e. we put certain constraints on the kind of environment in which we expect the system to work. Now in order to do such a *closed system analysis*, in addition to the controlling system we also need to model the environment.

In general there are digital and analog processes in the environment. Thus we need some way to model arbitrary analog behaviour. In [49], Maler, Manna and Pnueli introduced the notion of a *hybrid system*: hybrid systems are a formalism which allows the modelling of digital and analog behaviour in a common framework. As usual in computer science, the discrete part is modelled by an FSM. For the continuous part, real valued variables are used.

Figure 19 is once again the coffee machine, now modelled as a hybrid systems. As we can now specify some analog behaviour, we give a more detailed description of what is going on in the machine. Namely, we will describe how the water is boiled and the coffee (resp. tea) is poured into the cup. This is described by two continuous variables T (for temperature) and H (for height).

A hybrid system is a two phase system like a timed automaton (see Fig. 18). But while in a timed automaton, in the continuous phase only the clocks would increase with time, now our continuous variables can behave in any reasonable way as time passes. So in every location we need to define how our continuous variables will evolve. This is most easily done by using differential equations as usual in physics. Thus every location in the hybrid system has differential equations for each continuous variable. The functions describing the behaviour of the variables are called *activities*.

Further every location has an *invariant* – a formula over the values of the variables – which gives the admitted values for the variables. A process may only stay within a location as long as it does not violate the invariant.

The activities and the invariant of the start location are very simple: both variables will not change in the start state (the water is not heated and no water is poured into a cup), so their slope is zero (note that we use \dot{T} to indicate slopes in the figures, and T' in the running text). Further we may stay in the start state for any amount of time, so the invariant is just true.

As we will see further down, transitions may in fact be labelled in a much more complicated way than before. In the example however things have nearly not changed: every transition is labelled with a guard, an action and an assignment. The assignment however is not a mere reset any more, but can assign arbitrary values to the continuous variables.

Now the coffee machine behaves as follows: in the start location, the water keeps its temperature and no water is poured into the cup. We stay there until someone presses, say, the coffee button. Then we move to the

set to 20, as this is the initial temperature of the water. As soon as we enter the location, the water is boiled according to the law $T' = (120 - T)/20$. We are allowed to stay in this location until the water boils. For the coffee, we assume that a “boiling temperature” of 90 degrees is sufficient (while the tea really needs 100 degrees). Upon reaching the boiling temperature, the next transition, which delivers the cup, is taken. After the cup is delivered, the water is poured into the cup. As the water is not heated anymore, its activity⁸ is now $T' = 0$. The height of the water inside the cup will however now rise at two millimeters a second, so the activity of the height is $H' = 2$. The cup is filled until the water height is two centimeters, i.e. the cup is full (for the tea, we assume that different cups with a height of three centimeters are used). Then the system goes back to its start location and waits again for someone to push a button (maybe the tea button next time).

Let us go back from the example to the general definition of hybrid systems. As we have seen, a hybrid system is an FSM with a finite number of continuous variables. The behaviour of these variables is described by real-valued functions over time called *activities*. Time only passes while the system resides within a location. The activities of the variables may be different from location to location. An *invariant* – i.e. an arbitrary formula over the continuous variables – controls if the system is allowed to stay within a location.

Transitions occur instantaneously: they take no time. A transition has a *pre-* and a *post-condition*. The pre-condition is a formula over the variables: it must be fulfilled for the transition to be taken. After taking the transition, the variables may be set to new values, which are given by the post-condition. So the post-condition is an assignment to the variables, which however may depend on the previous values of the variables. Generally, an observable action is associated with a transition. An overview is given in Fig. 20.

Hybrid systems are a very natural way to model systems with mixed digital and analog components. They combine the modelling of discrete systems from computer science with classical continuous modelling from natural sciences as applied in control theory.

The semantics of a hybrid system can again be given by a valuation graph: here the valuations are the current values of the continuous variables. While delays are still labelled with the exact amount of time elapsed, the variables will change according to their activities. This is again the two phase characteristic of the transition system.

⁸ To keep things simple, we do not model the cooling down of the water.

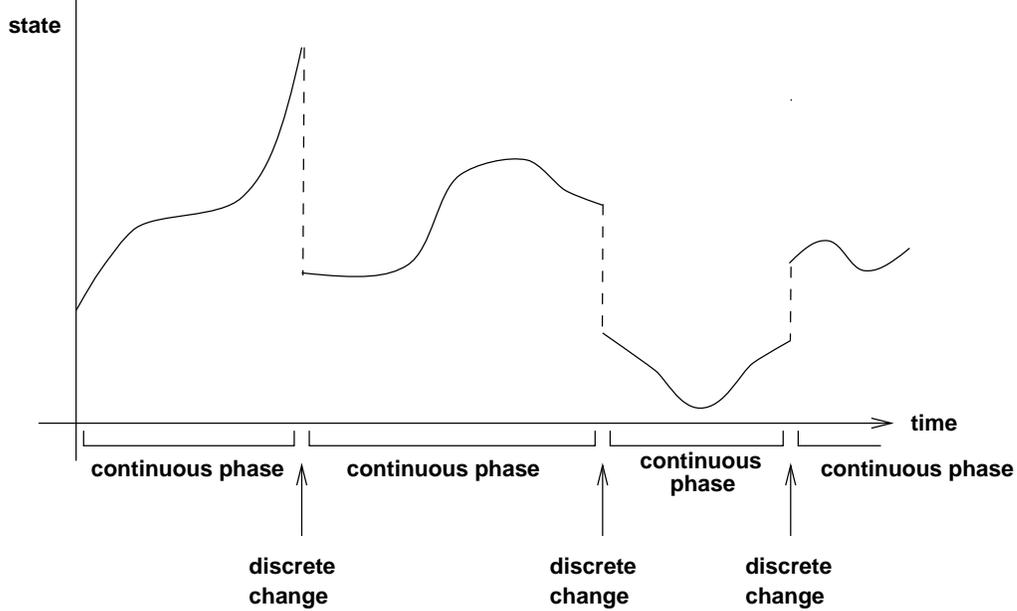


Fig. 18. Two Phase System

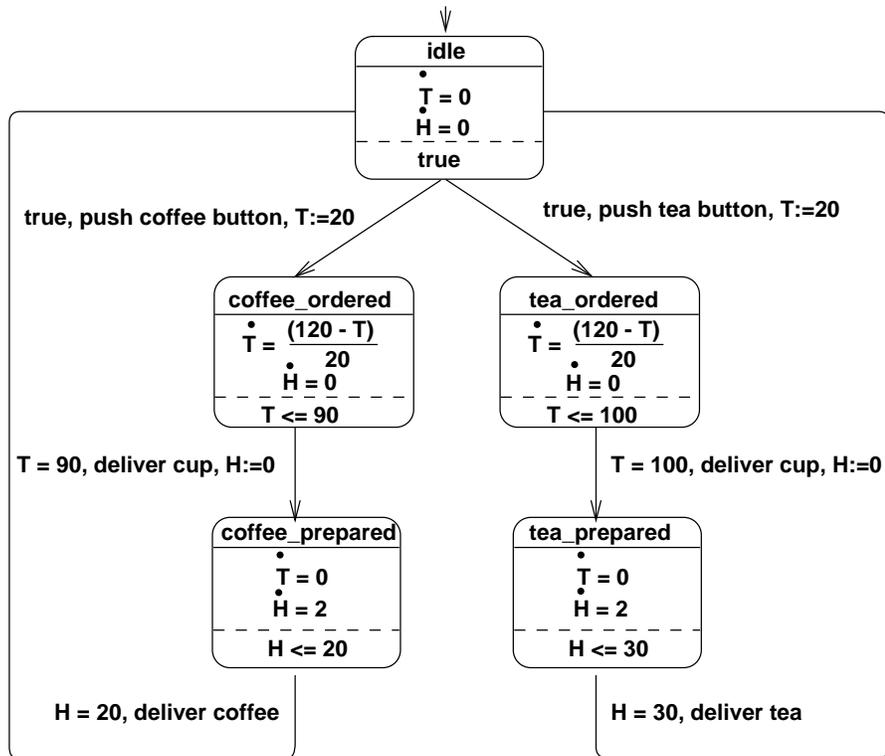


Fig. 19. A Hybrid Coffee Machine

The expressiveness of hybrid system is very rich: any physically realizable system should be describable as a hybrid system (and even some which are not realizable at all, as we allow arbitrary functions). However as usual there is a price for this expressiveness: the automatic analysis of hybrid system is only possible for very re-

stricted subclasses, and termination of the analysis algorithms cannot be guaranteed anymore. The general case is for sure quite intricate and it will probably take some time before methodologies are available here. However we have already seen two subclasses of hybrid systems which are automatically analysable: Timed automata are the

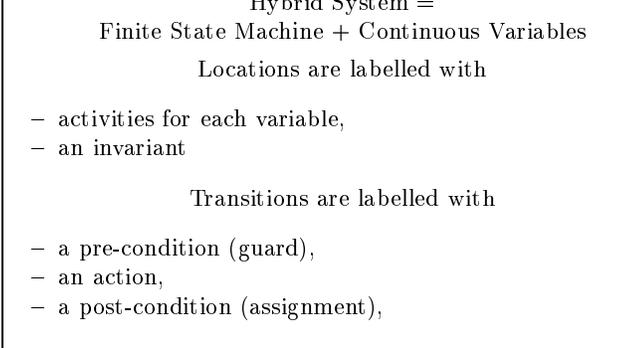


Fig. 20. What's in a hybrid system?

special case of hybrid systems where all activities have a slope of one, invariants and pre-conditions are comparisons of clocks (and clock differences) against natural numbers, and post-conditions are either to keep the value or to set it to zero. Drifting clock timed automata are a similar subclass, where the slope of the activities may vary within a given interval. However, there is a more expressive subclass which we will discuss in detail in the next subsection.

7.2 Linear Hybrid Systems

In 1993, Alur, Courcoubetis, Henzinger and Ho put forward the notion of *linear hybrid systems*, a subclass of hybrid system which can be analysed (semi-)automatically ([6]). In a linear hybrid system, invariants, guards and activities may only depend linearly on time or the values of variables. While this seems to be a severe restriction at first glance, case studies show that many interesting problems can be modelled as a linear hybrid system. The key idea to the analysis of linear hybrid system is what we call the *polyhedra technique*. We will explain the details of this technique in a separate subsection.

In order to give an easy example of a linear hybrid system, we simplify our specification of the coffee machine again. Instead of modelling the boiling of the water realistically, we use a linear approximation: we assume that the water's temperature rises by 2.3 degree every second. Thus we can model the machine as a linear hybrid system as done in Fig. 21. Note that only the activity $T' = (120 - T)/20$ had to be changed to $T' = 2.3$.

As with the region technique, the polyhedra technique allows a discrete representation of the infinite valuation graph of a linear hybrid system. However, in contrast to the case of timed automata, this representation need not be finite: if it is finite, the algorithmic analysis will terminate and yield a correct result, while it might be the case that the algorithm never terminates. This situation is called *semi-decidability* in computer science.

So the main result regarding linear hybrid systems is the existence of analysis algorithms. Thus there is the

of these algorithms cannot be guaranteed in general, in many practical cases a result will be produced. It is now a folk argument that in practice there is no real difference between an algorithm which always terminates or one which only sometimes terminates [77]. Note that for complicated systems, the analysis might take several days, and often it cannot be said a priori how long it will take. For the user, there is no difference in stopping an analysis after three days which might have terminated within a week or which might have run forever.

7.3 The Polyhedra Technique

While the approach to the analysis of timed automaton was based on regions, the analysis of linear hybrids systems is based on polyhedra. A polyhedron is a subset of the Euclidean space \mathbb{R}^n , which can be described by linear inequalities. So each guard and invariant of a linear hybrid systems describes a polyhedron.

If we choose linear hybrid systems as the subclass, then a polyhedron is a set describable by a linear formula. Assuming H and T are our continuous variables, the formula $0 \leq T \leq 100 \wedge H = 2T$ e.g. describes the set of all pairs $(x, 2x)$ where $x \in [0, 100]$.

Given a location p of a linear hybrid system and a polyhedron z , we can compute all points reachable from (p, z) in the same way as we did with the region technique. By $\text{succ}_\chi(p, z)$ we describe the set of points reachable by time passage, while $\text{succ}_e(p, z)$ are all points reachable from (p, z) by taking the transition e . The result of these operations is a pair (p', z') where p' is some location of the hybrid system and z' is again a polyhedron. Then reachability analysis can again be done by computing sets reachable by one step iteratively. The algorithm, which is analogous to the one based on regions, is given in Fig. 22.

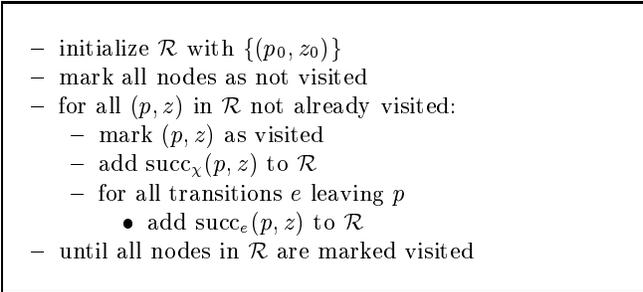


Fig. 22. Reachability Analysis in Linear Hybrid Systems

The main reason why this algorithm is correct is that when starting from some polyhedron, the successor states can again be described using polyhedra. However, termination is not guaranteed anymore, as there are generally infinitely many polyhedra.

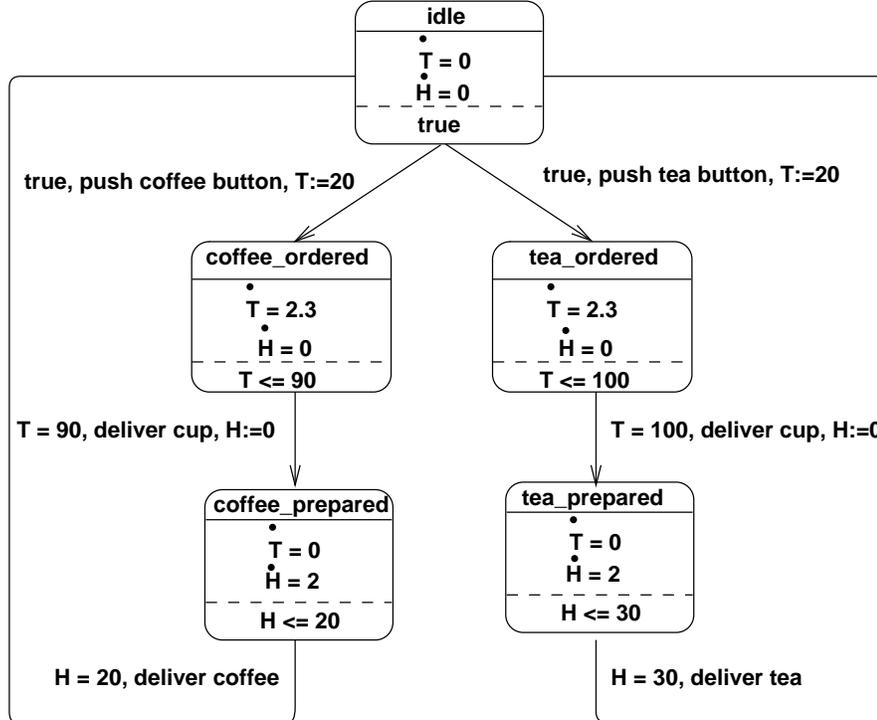


Fig. 21. Coffee machine as a linear hybrid system

7.4 Below and Above Linearity

As said before, timed automata are a proper subclass of linear hybrid systems. In principle, timed automata can be analysed using the polyhedron technique for linear hybrid systems, and in fact one can gain from this in speed. The good thing about timed automata is that the analysis is guaranteed to terminate.

There is even a larger subclass of linear hybrid system for which termination is guaranteed, which is the class of linear hybrid systems with a finite bisimulation. However, we cannot go into detail here but refer to [36].

However, if we go beyond timed automata into linear hybrid systems, then while there exist analysis procedures, their termination is not guaranteed anymore. When we cross the border of linearity and enter the field of non-linear systems, then in general we do not have any analysis algorithm at all. However, even non-linear systems can be analysed. Often, a linear approximation of the system can be found ([40]). There exist safe approximations, i.e. if the analysis proves the linear approximative to be well-behaved, then the original system is as well. Note that this is another quality of semi-decidability.

Approximations are also a good idea in the linear case if the analysis does not terminate. Then it might help to use over-approximations of polyhedra to force the algorithm to terminate.

This indicates that there is a vast class of hybrid systems which can be described and analysed using the tech-

niques of linear hybrid systems. The application in [74] clearly demonstrates the potential of applying HYTECH to real hybrid systems using approximations.

8 The Tools

- KRONOS
- UPPAAL
- HYTECH

In this issue of STTT, three tools for real time and hybrid systems are presented: KRONOS, UPPAAL and HYTECH. Each will be presented in an article of its own elsewhere in this special issue. Here we just give a short introduction to all of them.

KRONOS[28] is a model checker for timed automata. As input it takes a timed automaton and a TCTL formula. The output is a condition under which the automaton satisfies the formula. In the best case, the condition is just **true**, meaning the automaton will always satisfy the formula. If the formula is not satisfied, an error trace is generated.

KRONOS is mostly the work of Sergio Yovine, and was implemented as part of his thesis [82] at VERIMAG in Grenoble. It is now available in version 2.1a (September 1996). KRONOS has been successfully applied in a number of non-trivial case studies ([29, 30, 50]).

UPPAAL is also a model checker for timed automata. However the logic which is supported by UPPAAL is much simpler than TCTL, permitting more efficient algorithms. UPPAAL can deal with drifting clock timed automata and also allows integer variables in the system description, which is useful in many practical cases. UPPAAL has a nice graphical interface and a simulator which can be used to observe the system's behaviour.

UPPAAL is joint work of the Universities of Uppsala and Aalborg, initiated by Kim G. Larsen, Paul Pettersson and Wang Yi with Johan Bengtsson and Fredrik Larsson [19]. It was implemented in '94 and '95, and is now available in version 2.02. UPPAAL has proved useful even in industrial case studies ([34, 47]).

HYTECH is a tool for linear hybrid systems. As input it takes a description of a linear hybrid system and a set of analysis commands. These commands allow to program reachability analysis of linear hybrid system. HYTECH can be used to analyse all kinds of linear hybrid systems, so it can especially do all the analysis of which KRONOS and UPPAAL are capable, although sometimes less efficiently due to the general approach. Especially interesting is that with linear hybrid systems, it is easy to specify parametrized systems, and HYTECH will be able to synthesize parameter values, i.e. find the correct values for the parameters so that the system will work. Though sometimes one will run into the problem of non-termination.

HYTECH has been designed and implemented by Tom Henzinger, Pei-Hsin Ho and Howard Wong-Toi in 1995 at Cornell University[37, 12], but they have now moved to Berkely. HYTECH has been tested on a number of interesting case studies (see [38, 39, 42]).

All the forementioned tools are now established and well known in the formal methods and real time community.

9 Discussion

- history
 - advantages of continuous modelling

We have given an introduction into the field of continuous modelling of real time systems, presenting the notions of timed automata, drifting clock timed automata, linear hybrid systems and (general) hybrid systems. In Fig. 23 we give an overview of the "history" of continuous real time modelling as far as we have covered this field in the paper. This overview is intended to show how the theory of continuous real time modelling evolved over the last decade. Note the upcoming of the three tools presented in this issue of STTT in 1993, '94 and '95. Note further how the tools tackled the audio control protocol

case studies around.

Our presentation focused on the portion of theory of continuous model directly related to these tools. We could not account for the many other approaches to continuous real time modelling as e.g. the *duration calculus* [24, 25], ACP_ρ [15, 14] or *hybrid Petri Nets*[27, 69].

We would like to end the presentation with a short discussion of the usefulness and applicability of the models.

In Fig. 24 we give an overview of the various versions of coffee machines modelled in the paper: the untimed coffee machine, the timed automata coffee machine, the linear hybrid system coffee machine and the hybrid system coffee machine. As can be seen in this direct comparison, the FSM modelling the control of the systems does not change when adding time. Passing of time and continuous processes are modelled by augmenting the basic system with more and more specific information, getting a more and more realistic model of the coffee machine's behaviour. On the other side, the analysis of the system becomes more and more complex.

An important question arising in the early stage of the modelling process is the choice the right formalism. If there is a need to model continuous behaviour, then obviously the appropriate subclass of hybrid systems as presented here will be a correct choice. But there are even arguments for using the continuous approach in pure digital systems.

An important point is that sometimes in digital systems modelling one is forced to use a fixed global time scale. A major advantage of the polyhedra technique is that it will choose the appropriate time scale locally: if the next event cannot occur with the next three seconds, then three seconds is the "local" time scale, if events happen every millisecond, then milliseconds are the the "local" time scale.

Further compositional and refinement techniques work better with continuous than with discrete time: when combining two systems with a different time scale, both systems have to be altered so that they have an identical time scale before they can be combined in the discrete case, while in the continuous case there is no need for this⁹.

The development of the theory of timed automata and hybrid systems has proved the following important facts:

- continuous time models can be analysed automatically or at least semi-automatically despite their infinite and uncountable character
- the algorithms are not only of theoretical interest, but can be implemented in real tools

⁹ One could say that a continuously modelled system is already at the most fine grained time scale.

- 1988 Stankovič’s article on real time science [71, 72], Timed CSP [66]
- 1989 Difference Bound Matrices [31], Real Time Temporal Logic [8]
- 1990 Timed CCS [79], ATP [59], Model Checking for Timed Automata [3]
- 1991 Theory of Timed Automata [7]
- 1992 Safety Timed Automata [41], Hybrid Systems [49]
- 1993 KRONOS [82]
- 1994 HYTECH [37], Audio Control Protocol proved by hand [21]
- 1995 UPPAAL [19], HyTech and Uppaal solve Audio Control Protocol automatically [43, 46]
- 1996 Audio Control Protocol with Bus Collision solved automatically [18]
- 1997 Industrial Case Studies [34, 47]

Fig. 23. The “History” of Continuous Models

- these tools can be made efficient enough to solve real problems like the verification of Philips’ audio control protocol

10 Conclusions

- Improve Theory
- Improve Tools
- Improve Methodologies

Most of the microprocessors which you come across in your everyday life are not inside a computer: in fact they sit inside a lot of machines of every day life from cars to washing machine and video recorders. The development of real time and embedded systems is a more and more growing industry. While a malfunctioning video recorder is just a nuisance for the user – but maybe still a catastrophe for the manufacturer – errors in the software controlling your car’s anti skid system, an aeroplane or a power plant can have disastrous effects.

By using a good design methodology, a lot of errors can be detected and thus avoided during the design of software systems leading to better and less expensive software products. Thus good (and simple) design methodologies are obviously a desirable goal in the field of real time systems, probably one of the main goals in this field at the moment. Formal methods can be of great help in developing good and sound design methodologies. Therefore investing into the development of formal methods for real time systems is important for their emergence.

It is beyond the scope of this article to comment on all the relevant factors of designing these methodologies. However we would like to comment on just a few points which are closely related to the content of this paper.

The formalism of hybrid systems and timed automata has proved useful for real time systems over the last years, and hybrid methodologies are already used in industrial applications (see [76, 75, 74]). Despite this fact, we will point out some directions of possible future re-

search aiming at improving hybrid methodologies and their usability.

On the theoretical side, it would be interesting to see how far the limits of analysability can be pushed. Here a focus could be put on general algorithms to attack non-linear hybrid systems. For linear hybrid systems more space and time efficient methods are highly appreciated. Further hybrid systems should be compared to other continuous approaches (as e.g. *discrete event systems*[65, 64, 70]) in order to learn from and understand the relation between the different approaches.

On the tool side, the tools can hopefully be made even more efficient and user friendly. For efficiency, appropriate data structures for hybrid systems are still an issue. For user orientedness, it is important to find out what potential users expect from a real time tool. Right at the moment all tools concentrate mainly on the real time aspects, while users probably want a lot of things they know from “untimed tools”, like data types, control structures, modules, hierarchy, composition, abstraction and reusability.

On the practical side, there is a strong need to develop good and simple design methodologies for real time systems. These methodologies must address the following questions

- how to use the theory,
- how to use the tools (efficiently),
- what are the relevant issues for the industrial design process.

We have the strong hope that this issue of STTT will make people aware of the usefulness of hybrid methods for real time systems, and that this in turn will help in the development of the hybrid methodologies.

Web Pages: For your convenience we list the URL’s in the World Wide Web for the real time tools mentioned in this article in Fig. 10.

Acknowledgements. The authors would like to thank Luca Aceto for valuable comments during the making of this paper.

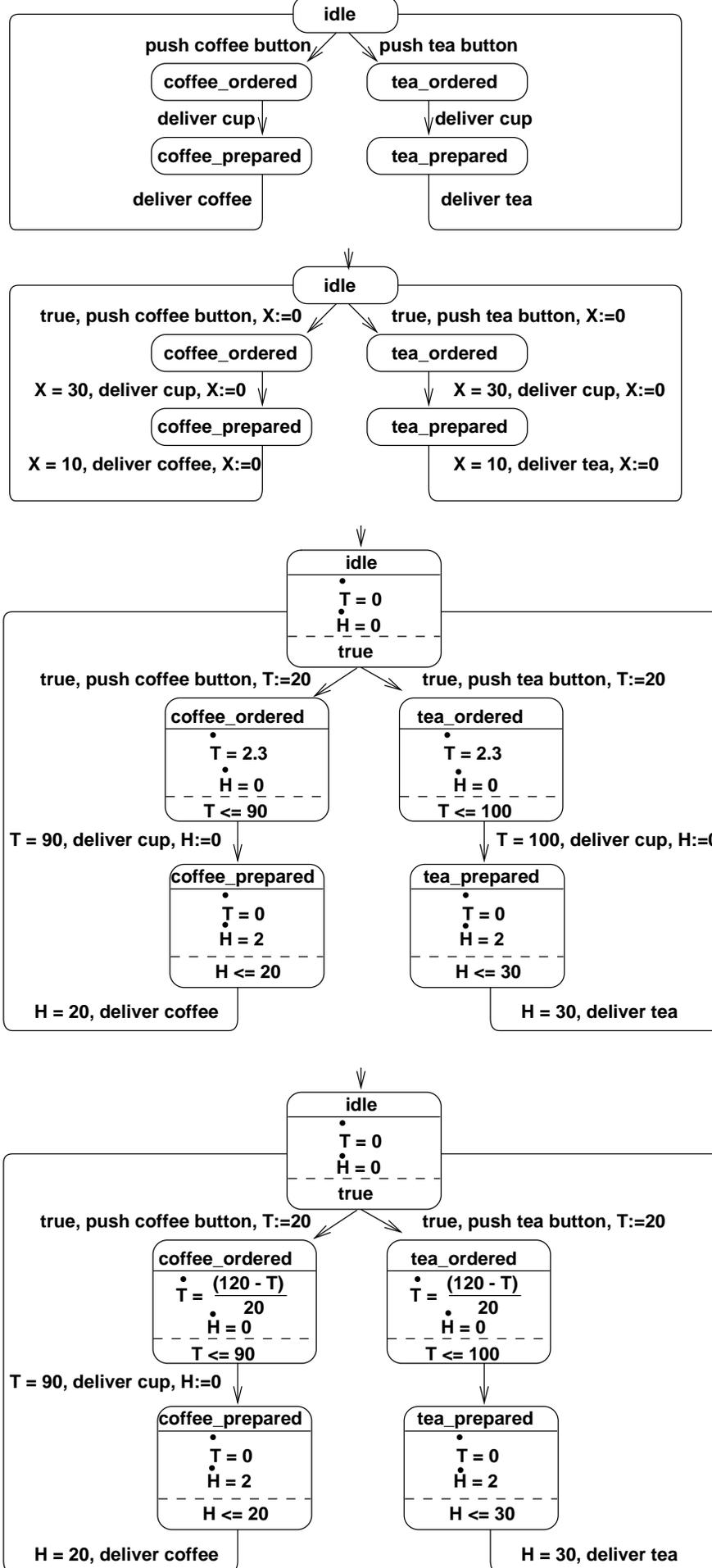


Fig. 24. Overview of the coffee machines

Fig. 25. URL's for the Tools

References

1. MISSING: Standard ML reference
2. R. Alur. *Techniques for Automatic Verification of Real-Time Systems*. Ph.D.-Dissertation, Stanford University, August 1991.
3. R. Alur, C. Courcoubetis, D. Dill. *Model-Checking for Real-Time Systems*. Information and Computation, May 1993, Vol. 104, No. 1, pp. 2-34.
4. R. Alur, D.L. Dill. *Automata For Modelling Real-Time Systems*. in: M.S. Paterson (Hrsg.), Proc.17th International Colloquium on Automata, Languages and Programming (ICALP), Warwick University, England, July 1990. Lecture Notes in Computer Science 443, Springer 1990, pp. 332-335.
5. R. Alur, C. Courcoubetis, T.A.Henzinger, N. Halbwachs, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, S. Yovine. The algorithmic analysis of hybrid systems. Theoretical Computer Science 138:3-34, February 1995.
6. R. Alur, C. Courcoubetis, T.A. Henzinger, P.-H. Ho. *Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems*. In Hybrid Systems I, Lecture Notes in Computer Science 736, Springer-Verlag, 1993, pp. 209-229.
7. R. Alur, D.L. Dill. *A Theory of Timed Automata*. in: Theoretical Computer Science Vol. 126, No. 2, April 1994, pp. 183-236.
8. R. Alur, T.A. Henzinger. *A Really Temporal Logic*. Technical Report No. STAN-CS-89-1267, Stanford University, Stanford 1989.
9. R. Alur, T.A. Henzinger. *Real-time Logics: Complexity and Expressiveness*. Proc. 5th Symp. on Logics in Computer Science (LICS'90), Philadelphia, PA., June 1990, pp. 390-401.
10. R. Alur, T. Henzinger. *Logics and models of real time: A survey*. In J.W. de Bakker, C. Huizing, W.P. de Roever, G. Rozenberg (eds.), Proc. REX Workshop "Real-Time: Theory in Practice", Lecture Notes in Computer Science 600, 74-106, 1992.
11. R. Alur, T.A. Henzinger. *Real Time System = Discrete System + Clock Variables*. in: STTT No. 1, November 1997, pp. ???-???
12. R. Alur, T.A. Henzinger, P.-H. Ho. *Automatic Symbolic Verification of Embedded Systems*. IEEE Transactions on Software Engineering 22:181-201, 1996. Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. Parametric real-time reasoning.
13. R. Alur, T.A. Henzinger, M.Y. Vardi. Parametric real-time reasoning. Proceedings of the 25th Annual ACM Symposium on Theory of Computing (STOC 1993), pp. 592-601
14. J.C.M. Baeten, J.A. Bergstra. *Real-Time Process Algebra*. Formal Aspects of Computing, 1991, Vol. 3, No. 2, pp. 142-188.
15. J.C.M. Baeten, J.W. Klop. *Process Algebra for Synchronous Communication*. Information and Control 60, 1984, pp. 109-137.
16. W.W.R. Ball. *A Short Account of the History of Mathematics*. http://www.maths.tcd.ie/pub/HistMath/People/Leibniz/RouseBall/RB_Leibnitz.html#CalculusDispute.
17. R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.
18. Johan Bengtsson, W. O. David Griffioen, Kre J. Kristoffersen, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, Wang Yi. *Verification of an Audio Protocol with Bus Collision Using UPPAAL*. In Proceedings of the 8th International Conference on Computer-Aided Verification. New Brunswick, New Jersey, USA, 31 July 31-3 August, 1996. Lecture Notes in Computer Science 1102, pages 244-256, R. Alur and T. A. Henzinger (Eds.).
19. Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, Paul Pettersson and Wang Yi. *UPPAAL - a Tool Suite for Automatic Verification of Real-Time Systems*. In Proceedings of the 4th DIMACS Workshop on Verification and Control of Hybrid Systems, New Brunswick, New Jersey, 22-24 October, 1995.
20. M. Bizga, J.-C. Fernandez, A. Kerbrat, L. Mounier. *Protocol verification with the Aldébaran toolset*. in: STTT No. 1, November 97, pp. ???-???
21. D.J.B. Bosscher, I. Polak, F.W. Vaandrager. *Verification of an audio control protocol*. In H. Langmaack, W.P. de Roever, and J. Vytupil, editors, Proceedings 3rd International School and Symposium on Formal Techniques in Real Time and Fault Tolerant Systems (FTRTFT'94), Luebeck, Germany, September 1994, Lecture Notes in Computer Science 863, pages 170-192. Springer-Verlag, 1994. Full version appeared in T. Rus and C. Rattray, editors, Theories and Experiences for Real-Time System Development, AMAST Series in Computing, Vol 2, pages 147-176, World Scientific, Singapore, 1994. Available as CWI technical report CS-R9445.
22. Alan Burns, Andy Wellings. *Real-Time Systems and Their Programming Languages*. Addison-Wesley, Wokingham, 1990.
23. K. Čerāns, J.C. Godesken, K.G. Larsen. *Timed Modal Specification - Theory and Tools*. in: C. Courcoubetis (Ed.), Proc. 5th Int. Conf. on Computer Aided Verification (CAV '93), Elounda, Greece, June/July 1993. Lecture Notes in Computer Science 697, Springer Berlin 1993, pp. 253-267.

- ration Specification for Shared Processors. in: J. Vytotil (Ed.), Proc. 2nd Int. Symp. on Formal Techniques in Real-Time and Fault-Tolerant Systems, Nijmegen, Netherlands, January 1992. Lecture Notes in Computer Science 571, Springer Berlin 1992, pp. 21-33
25. Z. Chaochen, C.A.R. Hoare, A.P. Ravn. *A Calculus of Durations*. Information Processing Letters, Vol. 40, No. 5, pp. 269-276.
 26. R. Cleaveland, J. Parrow, B. Steffen. *The Concurrency Workbench: A Semantics-Based Tool for the Verification of Concurrent Systems*. ACM Trans. on Prog. Lang. and Systems, Vol. 15, No. 1, Jan. '93, S. 36-72.
 27. R. David, H. Alla. *Petri Nets & Grafset - Tools for modelling discrete event systems*. Prentice hall, New York, 1992.
 28. C.Daws, A.Olivero, S.Tripakis and S.Yovine. *The tool Kronos*. In Hybrid Systems III, Verification and Control, Lecture Notes in Computer Science 1066, Springer-Verlag, 1996.
 29. C.Daws, A.Olivero, S.Yovine. *Verifying ET-LOTOS programs with KRONOS*. In Proceedings of the 7th IFIP WG G.1 International Conference of Formal Description Techniques FORTE'94, pages 227-242, Bern, Switzerland, October 1994. Formal Description Techniques VII, Chapman & Hall.
 30. C.Daws, S.Yovine. *Two examples of verification of multirate timed automata with KRONOS*. In Proceedings of the 1995 IEEE Real-Time Systems Symposium, RTSS'95, Pisa, Italy, December 1995. IEEE Computer Society Press.
 31. D.L. Dill. *Timing Assumptions and Verification of Finite-State Concurrent Systems*. in: J. Sifakis (Ed.), Automatic Verification Methods for Finite State Systems. Lecture Notes in Computer Science 407, Springer Berlin 1989, pp. 197-212.
 32. W.O.D. Griffioen. *Analysis of an Audio Control Protocol with Bus Collision*. Master thesis, Programming Research Group, University of Amsterdam, 1994.
 33. E. Harel, O. Lichtenstein, A. Pnueli. *Explicit Clock Temporal Logic*. Proc. 5th Symp. on Logics in Computer Science (LICS'90), Philadelphia, PA., June 1990, pp. 402-413.
 34. Klaus Havelund, Arne Skou, Kim G. Larsen and Kristian Lund. *Formal Modelling and Analysis of an Audio/Video Protocol: An Industrial Case Study Using UPPAAL*. Accepted for presentation at the 18th IEEE Real-Time Systems Symposium. San Francisco, California, USA, 3-5 December 1997.
 35. M. Hennessy, T. Regan. *A process algebra for timed systems*. Information and Computation, March 1995, Vol. 117, No. 2, pp. 221-239.
 36. Thomas A. Henzinger. *Hybrid automata with finite bisimulations*. Proceedings of the 22nd International Colloquium on Automata, Languages, and Programming (ICALP 1995), Lecture Notes in Computer Science 944, Springer-Verlag, 1995, pp. 324-335.
 37. Thomas A. Henzinger and Pei-Hsin Ho. *HyTech: The Cornell Hybrid Technology Tool*. In Hybrid Systems II, Lecture Notes in Computer Science 999, Springer-Verlag, 1995, pp. 265-294
 38. Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. *HyTech: the next generation*. Proceedings of the (RTSS 1995), pp. 56-65.
 39. Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. *A user guide to HyTech*. Proceedings of the First Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 1995), Lecture Notes in Computer Science 1019, Springer-Verlag, 1995, pp. 41-71.
 40. Thomas A. Henzinger and Pei-Hsin Ho. *Algorithmic analysis of nonlinear hybrid systems*. Proceedings of the Seventh International Conference on Computer-aided Verification (CAV 1995), Lecture Notes in Computer Science 939, Springer-Verlag, 1995, pp. 225-238
 41. T.A. Henzinger, X. Nicollin, J. Sifakis, S. Yovine. *Symbolic Model Checking for Real-time Systems*. Information and Computing, June 1994, Vol. 111, No. 2, pp. 193-244.
 42. Thomas A. Henzinger and Howard Wong-Toi. *Using HyTech to synthesize control parameters for a steam boiler*. In Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control, Lecture Notes in Computer Science 1165, Springer-Verlag, 1996, pp. 265-282.
 43. P.-H. Ho, H. Wong-Toi. *Automated Analysis of an Audio Control Protocol*. Proceedings of the Seventh International Conference on Computer-aided Verification (CAV 1995), Lecture Notes in Computer Science 939, Springer-Verlag, 1995, pp. 381-394.
 44. C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall, Englewood Cliffs, New Jersey 1985.
 45. F. Jahanian, A.K. Mok. *Safety Analysis of Timing Properties in Real-Time Systems*. IEEE Transactions on Software Engineering 12(9), September 1986, S. 890-904.
 46. Kim G. Larsen, Paul Pettersson, Wang Yi. *Diagnostic Model-Checking for Real-Time Systems*. In Proceedings of the 4th DIMACS Workshop on Verification and Control of Hybrid Systems, New Brunswick, New Jersey, 22-24 October, 1995.
 47. Magnus Lindahl, Paul Pettersson and Wang Yi. *Formal Design and Analysis of a Gear Controller: an Industrial Case Study Using UPPAAL*. Technical Report ASTEC 97/09, Advanced Software Technology, Uppsala University, 1 August, 1997.
 48. H.R. Lewis. *A Logic of Concrete Time Intervals*. Proc. 5th Symp. on Logics in Computer Science (LICS'90), Philadelphia, PA., June 1990, pp. 380-389. siehe auch: Technical Report TR-07-90, Harvard University, 1990.
 49. O. Maler, Z. Manna, A. Pnueli. *From Timed to Hybrid Systems*. In REX workshop Real-Time: Theory in Practice, Lecture Notes in Computer Science 600, Springer-Verlag, pp. 447-484, 1992.
 50. O.Maler, S.Yovine. *Hardware timing verification using KRONOS*. In Proceedings of the IEEE 7th Israeli Conference on Computer Systems and Software Engineering, ICCBSSE'96, June 12-13, 1996, Herzliya, Israel. IEEE Computer Society Press.
 51. Z. Manna, A. Pnueli. *The Temporal Logic of reactive and Concurrent Systems*. Springer-Verlag, New York, 1991.
 52. R. Milner. *Communication and Concurrency*. Prentice Hall, Englewood Cliffs, 1989.

- design*. In Foundations of Real-Time Computing: Formal Specifications and Methods. Kluwer Press, 1991.
54. J.C. Mulder. *The Inevitable Coffee Machine*. University of Amsterdam, Programming Research Group, Technical Report P8915, December 1989.
 55. R. De Nicola, M.C.B. Hennessy. *Testing equivalences for processes*. Theoretical Computer Science, 34(1-2):83-133, November 1984.
 56. X. Nicollin. *ATP: une algèbre pour la spécification et l'analyse des systèmes temps réel*. Ph.D.-Dissertation, Institut National Polytechnique de Grenoble, Grenoble 1992.
 57. X. Nicollin, A. Olivero, J. Sifakis, S. Yovine. *An Approach to the Description and Analysis of Hybrid Systems*. Proc. Workshop on Theory of Hybrid Systems, Lyngby, Denmark, October 1992, pp. 1-30.
 58. X. Nicollin, J. Sifakis. *An Overview and Synthesis on Timed Process Algebras*. in: K.G. Larsen, A. Skou (Eds.), Proc. 3rd Int. Workshop on Computer Aided Verification (CAV '91), Aalborg, Denmark, July 1991. Lecture Notes in Computer Science 575, Springer Berlin 1992, pp. 376-398.
 59. X. Nicollin, J.-L. Richier, J. Sifakis, J. Voiron. *ATP: an Algebra for Timed Processes*. In: Proc. IFIP TC 2 Working Conference on Programming Concepts and Methods, Sea of Galilee, Israel, April 1990, pp. 402-492.
 60. X. Nicollin, J. Sifakis, S. Yovine. *From ATP to Timed Graphs and Hybrid Systems*. Acta Informatica 30, 1993, pp. 181-202.
 61. A. Olivero, J. Sifakis, S. Yovine. *Using Abstractions for the Verification of Linear Hybrid Systems*. in: Lecture Notes in Computer Science 818, Proc. CAV '94, pp. 81-94.
 62. J.S. Ostroff. *Automated Verification of Timed Transition Models*. in: J. Sifakis (Ed.), Automatic Verification Methods for Finite State Systems. Lecture Notes in Computer Science 407, Springer Berlin 1989, pp. 247-256.
 63. J.S. Ostroff, W.M. Wonham. *A temporal logic approach to real time control*. In Proceedings of the 24th IEEE Conference on Decision and Control, pages 656-657, Florida, December 1985.
 64. P. J. Ramadge. *Control and Supervision of Discrete Event Processes*. PhD thesis, University of Toronto, Canada, 1983.
 65. P.J. Ramadge, W.M. Wonham, *On the supervisory control of discrete event systems*. The IEEE Proceedings, January 1989.
 66. G.M. Reed, A.W. Roscoe. *A Timed Model for Communicating Sequential Processes*. Journal of Theoretical Computer Science 58,1988, pp. 249-261.
 67. A.W. Roscoe. *Model-Checking CSP*. In A Classical Mind, Essays in Honour of C.A.R. Hoare. Prentice-Hall, 1994.
 68. M. Schiebe, S. Pferrer (Hrsg.). *Real-Time Systems Engineering and Applications* Kluwer Academic Publishers, Boston 1992.
 69. S. Schöf, M. Sonnenschein, R. Wieting. Efficient Simulation of THOR Nets. In G. De Michelis, M. Diaz (Eds.), Prod. 16th International Conference on Application and Theory of Petri Nets, Turin, Italy, June 1995, Lecture pp. 412-431.
 70. R.S. Sreenivas, B.H. Krogh. *On Condition Event Systems with Discrete State Realizations*. In Discrete Event Dynamic Systems 1, 1991, pp. 209-236.
 71. J. Stankovic. *Misconceptions About Real-Time Computing*. IEEE Computer Vol. 21, No. 10, 1988, pp. 10-19.
 72. J.A. Stankovic. *Real-Time Computing Systems: The Next Generation*. in: Hard Real-Time Systems (Tutorial). IEEE Computer Society Press, Washington D.C., 1988, pp. 14-38.
 73. J. Stankovic, K. Ramamritham. *Introduction*. in: Hard Real-Time Systems (Tutorial). IEEE Computer Society Press, Washington D.C., 1988. pp. 14-38.
 74. T. Stauner, O. Müller, M. Fuchs. *Using HYTECH to Verify an Automotive Control System*. in: Hybrid and Real Time Systems, Lecture Notes in Computer Science 1201, Springer Berlin 1996, pp. 139-153.
 75. P. Terwiesch, E. Scheiben, A.J. Petersen, T.Keller. *A Digital Real-Time Simulator for Rail-Vehicle Control Systems Testing*. in: Hybrid and Real Time Systems, Lecture Notes in Computer Science 1201, Springer Berlin 1996, pp. 199-212.
 76. C. Tomlin, G. J. Pappas, J. Lygeros, D. Godbole, S. Sastry. *A Next Generation Architecture for Air Traffic Management Systems*. in Hybrid Systems IV, Proceedings of the 4th International Conference on Hybrid Systems, Lecture Notes in Computer Science, published by Springer-Verlag, Ithaca, NY, October 1996
 77. P. Wolper. *A unifying approach to exploring infinite state spaces*. Talk given at the the summer school on infinite systems, Grenoble, 1997.
 78. P. Wolper. *The meaning of "formal"*. Strategic Directions in Computing Research Formal Methods Working Group Position Statement, <http://www.montefiore.ulg.ac.be/~pw/sdcr/formal-methods-surveys.html>
 79. W. Yi. *Real-time behaviour of asynchronous agents*. in: Proc. CONCUR '90, Amsterdam, The Netherlands, August 1990. Lecture Notes in Computer Science 458, Springer New York 1990, pp. 502-521.
 80. W. Yi. *A Calculus of Real Time Systems*. Ph.D.-Dissertation, Chalmers University of Technology, Göteborg 1991.
 81. W. Yi. *CCS + Time = an Interleaving Model for Real-Time Systems*. Proc. 18th Int. Coll. on Automata, Languages and Programming (ICALP), Madrid, July 1991. Lecture Notes in Computer Science 510, Springer New York 1991, pp. 217-228.
 82. S.Yovine. *Methodes et outils pour la verification symbolique de systemes temporises*. Ph.D. Thesis, Institut National Polytechnique de Grenoble, France, May 1993.