CITIUS, VILIUS, MELIUS
GUIDING AND COST-OPTIMALITY IN MODEL CHECKING OF TIMED AND HYBRID SYSTEMS

by

ANSGAR FEHNKER

ISBN 90-9015716-6
IPA Dissertation Series 2002-08

Typeset with LaTeX2e.
Printed by Print Partners Ipskamp, Enschede.
Cover by Aimée Terburg, Groningen.

The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics). The research reported in this dissertation has been supported by the Netherlands Organization for Scientific Research NWO under contract SION 612-14-004.

CITIUS, VILIUS, MELIUS
GUIDING AND COST-OPTIMALITY IN MODEL CHECKING OF TIMED AND HYBRID SYSTEMS

EEN WETENSCHAPPELIJKE PROEVE OP HET GEBIED
VAN DE NATUURWETENSCHAPPEN, WISKUNDE EN INFORMATICA

Proefschrift

TER VERKRIJGING VAN DE GRAAD VAN DOCTOR
AAN DE KATHOLIEKE UNIVERSITEIT NIJMEGEN,
VOLGENS BESLUIT VAN HET COLLEGE VAN DECANEN
IN HET OPENBAAR TE VERDEDIGEN OP
MAANDAG 15 APRIL 2002
DES NAMIDDAGS OM 1.30 PRECIES
DOOR

Ansgar Fehnker

GEBOREN OP 27 JULI 1971 TE TEGLINGEN BIJ MEPPEN
DUITSLAND

Promotor:

Prof.dr. Frits W. Vaandrager

Manuscriptcommissie:

Prof.dr. Rajeev Alur, University of Pennsylvania, VS
Dr. Oded Maler, CNRS-Verimag, Frankrijk
Prof.dr. Wang Yi, Uppsala Universitet, Zweden

Preface

Writing a thesis is an ordeal, writing a preface even more so. It is well known that almost nobody is going to read the thesis (except for the manuscript committee, my supervisor, Judi Romijn, Angelika Mader, Jim Kapinski and maybe you, honorable fellow scientist. Thanks!), but almost everybody is going to read the preface. Probably, because the reader hopes to get some insight into the author's mind, to see who else was involved, to see whether their own name is mentioned, or to see who is responsible. Or maybe just because this is the most readable part of this book.

Nicht nur der Tradition gehorchend möchte ich mich als Erstes bei meinem Doktorvater Prof. Dr. Frits Vaandrager bedanken. Ohne ihn wäre dieses Buch nicht geschrieben worden, auf jeden Fall nicht durch mich. Aber Dank gebührt ihm nicht nur, weil es ihm behagt hat einem Mathematiker die Umschulung zum Informatiker an zu bieten, er hat mich all diese Jahre hindurch auch noch begleitet. Er ist ein kritischer Geist, der mich gelehrt hat, dass der Teufel nur allzu häufig im Detail steckt. Es geht nicht nur darum, dass etwas stimmt, es sollte auch bewiesenermaßen wahr sein. Frits ist aber auch ein äußerst angenehmer Chef, der es versteht eine stimulierende Arbeitsatmosphäre zu schaffen.

As you take a look on the first pages of each chapter (page 19, 43, 67, 95, and 109) you will see that most of them are a result of a joint effort. I therefore like to thank my co-authors Gerd Behrmann, Ed Brinksma, Thomas Hune, Kim Larsen, Angelika Mader, Paul Petterson, Judi Romijn and Frits Vaandrager, since they made – believe it or not – doing science a fun experience. To get an impression you should try to use Danish feta cheese to illustrate 3-dimensional regions. This works much better than mozzarella. I was lucky to participate in the EU research project VHS. Not only did it lead to most of the joint work in this thesis, it was also an outstanding opportunity to meet many interesting researchers from all over Europe. I enjoyed this project a lot. Thanks. I would also like to thank Henning Dierks, though our joint work is not included in this thesis.

Besonders herzlich möchte ich mich auch bei meiner Doktormutter Mirèse Willems bedanken. Sie hat nicht nur dafür gesorgt, dass der Laden lief, sondern hatte auch immer ein offenes Ohr für die Nöte der Doktoranden.

Erg bijzonder was dat ik, ook al moest ik de afgelopen jaren meerdere keren binnen ons mooie gebouw verhuizen, een vaste kamergenote had, waar ik het erg

goed mee heb kunnen vinden. Mariëlle nam altijd de tijd, ook als ze zelf druk bezig was, om mijn vaak onnozele vragen over wiskunde, informatica of het gebruik van de Engelse taal te beantwoorden. Ook kon ik altijd bij haar terecht voor stroopwafels, chocolade (Aldi rules) of letters (Een van de meest nijpende problemen in de moderne wiskunde is het gebrek aan letters). En niet te vergeten dat je met Mariëlle ook de importante zaken van het leven kunt bespreken, zoals de lokale middenstand, het openbaar vervoer, of de nieuwe outfit van Prinses Marilène.

If you ever have the opportunity to join the Informatics for Technical Applications group, you should definitely try it. It will give you the opportunity get a better insight in various fields such as European computer science human interest stories, Danish Christmas traditions, French cuisine and automotive transportation, Dutch immigration service, European royalty, North German verbal communication, South German feminism and welding, Brabantian proverbs and naming schemes, including the joy of having multiple initials. Thanks for all those cozy cups of coffee.

Via deze weg wil ik nog graag Marieke (A2rechts) en Karin (B1) bedanken. Marieke heeft me in Nederland geïntroduceerd, en Karin heeft me bij Melanie geïntroduceerd. Beide zijn erg goed bevallen, en ik ben er erg dankbaar voor. Verder wil nog ik nog Mies en Mowie bedanken, omdat zij me door de jaren gesteund hebben, en altijd verheugd waren als ik rond (hun) etenstijd weer thuis kwam.

It all started, of course, when I grew up in Teglingen, Lower Saxony; my paranymph Claudia already shared the frontmost bench in elementary school with me, since we were the shortest. But as I had many schoolmates, numerous teachers, a lot of neighbors, and an extensive family – which I still have – I omit special thanks. Let me just say: Those years didn't harm, thanks.

The reader who is just reading the preface is advised to take a look at the cover, which was designed by Aimée, and for those who intend to read the remaining 154 pages I will conclude with a proverb in my mothers tongue, which captures the basic feeling when you are writing a thesis.

Kien Tied, kien Tied.

Low Saxon proverb

Contents

Tables and Figures

1

Introduction

The 9000 series is the most reliable computer ever made. No 9000 computer has ever made a mistake, or distorted information. We are all – by any practical definition of the word – foolproof, and incapable of error.

HAL 9000, 2001: A Space Odyssey

1.1 Formal Methods

Tuesday, June 4, 1996, marks the beginning of a new chapter for the European aerospace industry. The floodlit Ariane 5, with its central fat rocket and its two thin solid boosters aside, awaits its maiden flight. Since the visibility criteria are not met at the beginning of the launch window at 08.33h local time, the launch is postponed. The electric field at the launch site of Kourou is negligible; there is no risk of lightning. Within the next hour the weather conditions improve, such that the rocket ignites for lift off at 09.33.59h. It gains height quickly. About 40 seconds after lift off the launcher suddenly veers from its intended path. The links between the boosters and the main stage break because of high aerodynamic load. This triggers the self-destruction mechanism. The debris of the vessel scatter over an area of about 12 square kilometers of the surrounding mangrove swamps.

According to the report of the Inquiry Board [Lio96] the vehicle veered from its path because the subsystem that computes the velocity and angle of the launcher was malfunctioning. The malfunctioning subsystem was backed-up by a second identical system, but this failed for the very same reason. The error originated in a function that worked perfectly well in Ariane 4 for years. The horizontal velocity of Ariane 5 in the early part of its trajectory, however, is much higher than that of Ariane 4. This caused a software exception when this data was converted from one format to another. Bitter was that the function that caused the error served no purpose anymore in the Ariane 5; it was maintained for reasons of commonality.

The design of the Ariane 5 was biased towards managing random failures; for this it included for example a back-up system for each critical system. However, the destruction was not caused by a random failure, but by an error in the design. The destruction of Ariane 5 now serves as one of the standard examples to moti-

vate the use of Formal Methods. The purpose of these methods is to help making a good design, and to find design errors way before the system is implemented. These methods have proven to be useful in numerous case studies [CW96]. Formal Methods use mathematically based languages to describe soft- and hardware systems, and provide methods and techniques to prove whether a given model does satisfy the requirements. The languages serve for modeling systems unambiguously, which is itself valuable and does in many cases reveal inconsistencies.

The formal model then allows also to employ techniques to prove mathematically that the model is correct. This is process is called verification. Verification can be done either by proving that the model relates to a simpler correct model, or by proving that the model satisfies a correctness property. These proofs can be done manually, but even for small systems they tend to be tedious and error prone. The two most prevalent methods techniques to computer aided verification are *Theorem Proving* and *Model Checking*. Theorem Proving automates the process of proving correctness, but requires in many cases user interaction. Model checking can handle only restricted classes of models, but if it is applicable, it verifies the correctness automatically, or it finds an error and gives information on how to reproduce it. In this thesis we will study model checking for timed and hybrid automata models.

1.2 Timed and Hybrid Automata

A formalism that is widely accepted for modeling soft- and hardware systems is that of *automata*[1], or *labeled transition systems*. An automaton describes the behavior of a system as a set of transitions from one state to another. A state can be considered as a snapshot of a system. The state of mechanical cash register for example represents the position of the wheels in the clock work, or, on a more abstract level, what can be seen on the display. Whenever a wheel turns or the display changes we may model this as a transition from one state to another. We may label this transition with a symbol, to model that the state changes by pressing the key with that symbol.

Automata have proven to be suitable for modeling in particular *reactive* and *concurrent systems*. As the name suggest, reactive systems interact with their environment, which is usually a process that has to be controlled. Based on input from this process a reactive system gives feedback to that process, to establish that

[1] The term automaton may cause confusion to those unfamiliar with it. There is nothing automatic in an automaton. An automaton is a description of a system, rather than the system itself; the term automata is also used as synonym for the language of the description. The philosophy of an automaton model is to consider the system as if it were implemented as a physical automaton. This makes this language in particular suitable for modeling soft- and hardware systems.

Figure 1.1: The train gate has to be closed before the train passes the gate.

the overall system behaves as desired. A typical example is a train gate controller, which closes the gate when the sensor detects an approaching train. The correctness of a reactive system depends strongly on its environment; verification has also to consider the process that has to be controlled. As illustrated by the Ariane 5, a correct controller for one rocket, may behave incorrectly in another.

In many cases subsystems have to interact with other subsystems to complete a task. These systems are called concurrent systems. Most approaches based on automata have a notion of synchronization that allows to model subsystems separately. A typical example of a concurrent system is a air travel booking system, which allows different travel agents to issue tickets concurrently.

Some systems act in an environment in which time plays an important role. The train gate controller for example has to close the gate within some seconds, rather than hours, and the booking system should acknowledge a request within a certain time. Timed automata include so called *clocks* – continuous variables with rate one – to capture timing aspects. In some cases clocks alone are not sufficient for modeling the continuous behavior of a system, for instance, if the model has to take the early trajectory of a rocket into account. For this there is the class of

hybrid automata that allows to put any type of continuous behavior into the system description[ACH+95]. Even though hybrid automata are richest of these classes in terms of expressivity, when it come to model checking timed automata or untimed automata may be a better choice.

1.3 Model Checking

Given a formal model one aims to show that it satisfies its specification. This specification might be given by a simpler model, or, as in this thesis, by a formula in a temporal logic. These logics allow to specify properties such as "Each request will eventually be acknowledged", or "Whenever a train crosses, the gate is closed." Actually, this thesis deals only with simple reachability problems, i.e. whether a given state can be reached from the initial configuration. Usually these states are considered bad states which should not be reachable. Take as example a state in which a train is on the crossing and the gate is open, or a state in which two agents issued a ticket for the same seat.

The philosophy of model checking is straightforward. To show that a state is not reachable the model checker searches all states exhaustively. If it finds a bad state, i.e. one that indicates that the model does not satisfy the specification, it will produce a counterexample that helps to find an error in either the design or the model. A prerequisite for this approach is that the model is finite, which is the case if the number of states and transitions are finite. In the case of timed and hybrid automata each valuation of the real valued variables leads to a state, and hence to an infinite and uncountable number of states.

The solution to this problem is to consider sets of states rather than individual states. These sets are called symbolic states. In the case of timed automata it is possible to give a partition of the state-space into a finite number of symbolic states [AD94]. The model checking algorithm is guaranteed to terminate in any case; we say that the reachability problem for timed automata is decidable. For the richer class of hybrid system there is no general decidability result. As matter of fact it has been shown that the reachability problem is undecidable. But, the situation is not a bad as it seems. For some sub-classes of hybrid automata decidability has been proved [HKPV95]. In addition, even if it cannot be guaranteed that the model checking algorithm does terminate in general, for many instances of a hybrid system it does terminate, and can thus give valuable information.

The main problem of model checking is known as the state-space explosion problem. This refers to the fact that the number of (symbolic) states in a model tends to grow exponentially with the number of components. Even for a relative small number of components, model checking may fail due to a limited amount

of memory. Approaches that proved to be able to tackle, or at least soften, this problem are the use of compact data structures, exploitation of symmetry, abstraction, or hashing, to name a few. As a consequence, model checking is able to aid with the design of soft- and hardware systems and is becoming widely accepted as verification technique in industry.

1.4 The Scope of this Thesis

This thesis does not deal with typical verification problems. Most of this thesis deals with finding feasible or optimal solutions, rather than errors. As mentioned before, timed automata are suitable for modeling the timing aspects of a system. Timed automata, however, can also be used for modeling scheduling problems. Similar to the booking system, we can for example model several agents that can share information via a telephone network. Suppose that each of then owns a distinct piece of information, and that they agree on a protocol on how to exchange information.

A typical verification problem would be to prove that a one-to-one communication cannot be disturbed by a third party. In contrast, a static scheduling problem[2] would be to find a schedule such that all agents know eventually all information. Given a timed automaton model, one can use a model checker to search for a state in which all agents have complete information. The "counterexample" then serves as a schedule.

Finding a feasible schedule is often not the primary problem. Rather, the problem is to find an optimal or fairly good schedule. A good schedule for the agents in the telephone network might minimize the number of calls, the time, or the cost of the telephone bill, maybe under the assumption that the agents use different rate plans. The model checking algorithm, however, has no notion of a "good" or "optimal" counterexample. In verification each counterexample is as bad, since it points to an error. If we want to use a model checker to find optimal schedules it is necessary to introduce a notion of cost, and associate a cost with each error state (as they represent feasible solutions). The model checking algorithm has then to be modified to search for the optimal solution, rather than for any solution. It not obvious that the modified model checking does terminate in any case, since the cost of a state is basically unbounded. The optimal solution may not be computable. In addition, introducing a notion of cost may necessitate to either modify exiting data structures or to develop new ones.

[2] This problem is called static, since the setting does not change while the schedule is executed. *Dynamic* scheduling considers problem that may change during the course of the schedule due to changing demands.

The approach to use model checking for scheduling suffers, just as model checking for verification, from state-space explosion. But since we are looking for solutions rather than errors, we do not need to explore the full state-space. As soon the model checker finds a solution, it can stop the exploration of states that cannot lead to better solutions. The search space may be reduced further, if the model checker searches first parts of the state-space that are likely to contain a good solution. The scheduling problem is often such that one can point at transitions that are likely to contribute to or to foil a good solution. The model checker should then prefer or postpone exploration of that transition.

Guiding the model checkers to promising parts of the state-space can also useful for verification. Despite of the efforts to mitigate the state-space explosion problem, there are interesting problems that are too large to be model checked. Nevertheless, in many cases one can identify components and transitions that have to be involved in order to possibly violate a property. Guiding allows us to explore first the part that is likely to contain an error. Of course, this mean that the model might not be completely verified, but it helps in debugging the design, even if a complete exploration is impossible. Guiding may also have a positive effect on the size of the state-space, provided that the model checker uses a symbolic state representation. It has been observed that the number of symbolic states depends on the order in which they are explored. Guiding the exploration allows us to utilize this effect. Even though the guiding the state-space exploration is mostly motivated by scheduling problems, its results are also valuable for general verification problems.

This thesis shows how to model scheduling problems as timed automata models, extends the timed automaton model with a notion of cost, shows that the minimal-cost reachability problem is computable, presents efficient data structures, and sketches how to guide the state-space exploration. It then presents a way to over-approximate sets of states for a sub-class of hybrid automata and investigates whether search order can influence the size of the state-space positively. This thesis is about model checking, and spends not only attention to the algorithmic part, but also to modeling. It discusses the model of a part of a steel plant in great detail, and considers furthermore models that involve people with different physical abilities, planes, experimental batch plants, a router, a communication protocol and cars.

1.5 This Thesis: Context and Overview.

Most of the research that led to this thesis is a result of the participation of the University of Nijmegen in the VHS project[3]. The VHS project (Verification of Hybrid Systems) is a European Union Esprit long term research project, and was

[3] European Union Esprit-LTR Project 26270 VHS (Verification of Hybrid Systems)

set up to improve the take-up of modern information technologies in industry. One of the major objectives of the project is to analyze a number of case studies of the industrial partners. This includes explicitly the use of existing verification tools for timed and hybrid systems.

Most of the case studies in the VHS project have some scheduling aspects. The motivating example of this thesis is to schedule a part of the Sidmar steel plant, which is also know as case study 5. Using the timed automata model checker UPPAAL it was possible to generate feasible schedules. These first experiments with this approach in the VHS project led to an extensive cooperation with the UPPAAL team, as can been seen from the list of co-authors. Hence, most models that are presented in this thesis use the UPPAAL input language of networks of timed automata.

The thesis is organized as follows. Chapter 2 shows how to model scheduling and planning problems as timed automaton model. It considers the general class of job shop problems and describes the Sidmar plant model in more detail. Boel and Stremersch presented a Petri-Net model of this case study in [BS99]. Yovine and Niebert used the timed automata model checker KRONOS to generate schedules for case study 1 of the VHS project [NY99]. Brinksma and Mader used the untimed model checker SPIN to compute schedules for a Promela model of case study 1 [BM00].

Chapter 3 introduces *Linearly Priced Timed Automata* (LPTA), an extension of the timed automaton model with cost. The cost might increase with a fixed rate as time passes, and by a fixed amount if a transition is taken. The cost rate may change on taking a transition. A basic notion of a symbolic state extended with cost allows to prove that the optimal solution is computable. Similar and independent work has been presented by Alur et al. [ATP01].

The data-structures that are used in Chapter 3 are guaranteed to be inefficient. Chapter 4 introduces *Uniformly Priced Timed Automata* (UPTA), which differ from LPTAs in that the cost rate is constant and may not change on taking a transition. This class covers for example minimum time optimality. The chapter presents an efficient data structure for UPTAs, a class of optimal search orders, and modifications of the algorithm that allow manual guiding of the exploration. A number of experiments illustrates the results. The minimum time reachability problem, and the more general problem of controller synthesis has been solved in [AM99]. Niebert et al. present an alternative solution to the minimal time problem in [NTY00]. Abdeddaïm and Maler show in [AM01] that the model checking algorithm can be further tailored to solving job shop problems. Reffel and Edelkamp [RE99] presented an modified model check algorithm for untimed systems, that allows to guided the state-space exploration in order to find an error state.

Chapter 5 presents an efficient data structure for the full class of linearly timed

priced automata. This approach uses linear programming to realize the necessary operations on symbolic states. The main application example is a aircraft landing problem taken from [BKA00]. The results confirm that the prototype is able to compete with other approaches, such as linear programming based tree search.

For LPTAs we have developed a representation of sets of states that guarantees termination. There is no equivalent for general hybrid systems. Chapter 6 proposes for a sub-class a notion of symbolic states based on bounded polyhedra. We illustrate for two cases studies that this representation allows to derive automatically outer bounds on the set of reachable states. Similar approaches are presented in [CK99], [Dan99] and [Var98]. The foregoing chapters show that heuristic search allows to find a (close to optimal) solution quickly. As complement to this result, the remainder of Chapter 6 investigates whether a complete state-space exploration can benefit from heuristic search orders. Chapter 7 reviews the content of this thesis, gives some general remarks on experimental research in the field of formal methods and concludes with directions for future research.

1.6 Bibliographical Notes

All chapters in this thesis are based, at least to some extent, on earlier publications. Chapter 2 is based on [Feh99], that introduces a timed automaton model of the Sidmar steel plant, and [Feh00a], that sketches how to move from a reachability to a scheduling algorithm. Chapter 3 is based on [BFH+01b], and Chapter 4 on [BFH+01a]. The latter includes also experimental findings that were presented in [BMF02]. The content of Chapter 5 is based on [LBB+01]. Finally, Chapter 6 is based on [Feh98] which presented a polyhedral approximation technique for hybrid systems, and [Feh00b] which reported on experiments with heuristic search orders. Work on timed automata semantics for PLC-automata [DFMV98], a formalism for modeling industrial applications that use PLCs to control a process, has not been included in this thesis.

Each chapter has its own introduction. These introductions contain references to essential work in the context of that particular chapter, that were omitted in this introduction. A general overview of formal methods an model checking was given by Clarke and Wing in [CW96]. Alur and Dill introduced the timed automata model in [AD94]. Common model checkers for timed automata are KRONOS [Yov97], and UPPAAL [LPY97, ABB+01]. Alur et al presented a framework for algorithmic analysis of hybrid systems in [ACH+95]. An introduction to job shop scheduling was given by French in [Fre82], and a recent overview by Jain and Meeran in [JM99].

2

From Schedulability to Reachability

2.1 Introduction

Case study 5 of the VHS project (CS5) is brought into the project by Sidmar, a flat steel producer from Ghent, Belgium. It deals with the part of an integrated steel plant where molten pig iron coming from the blast furnace, is converted into steel of different qualities before it enters the hot rolling mill. The raw iron enters the system is batches of about 290 ton. Each batch is transported in a so called *ladle*. Depending on the quality of steel that has to be produced, a batch moves along the different machines, to undergo the necessary treatments. Finally, the steel is casted in the rolling mill, where the flat steel is produced.

This problem might look, at first sight, similar to a job shop scheduling problem. In job shop scheduling theory one supposes that there are a number of jobs and machines. A job is usually defined as a sequence of operations which have to be executed in a given order. Each of these operations is performed by a particular machine for a given period of time. It is also assumed that each machine can perform only one operation at the same time [CP89]. Furthermore, one often assumes that each job can perform on each machine only once an operation. The problem is to schedule the operations in a way that it minimizes the time it takes to complete all jobs.

Most assumptions mentioned in the previous paragraph do not hold for case study 5. Some operations may be performed by different machines, other resources may be used by two jobs concurrently. The duration of some operations is not fixed but lies in an interval. There is a lot of freedom in how to schedule the jobs. On the other hand we have non-trivial constraints. Due to the topology of the plant there

This chapter contains excerpts form the following publications:

[Feh99] A. Fehnker. *Scheduling a Steel Plant with Timed Automata.* Sixth International Conference on Real-Time Computing Systems and Applications (RTCSA'99), 1999.

[Feh00a] A. Fehnker. *Bounding and Heuristics in Forward Reachability Algorithms.* CSI report CSI-R0002. *2000.*

are operations on certain machines that prevent operations of other jobs on other machines. These machines are not accessible during that operation. There are also resources that move, jobs that cannot wait indefinitely for a machine to become free and other deadlines. In this context it can even be difficult to decide whether a feasible schedule exists.

In this chapter we want to show how to reformulate scheduling problems as a reachability problem that can be solved by a verification tool. The timed automata [AD94] based modeling languages of the verification tool UPPAAL serves in this approach as the basic input language to describe the scheduling problem. The reachability algorithm is then used to find a state in which all jobs are completed. Firstly, we consider the class of job shop problems. Since it is a well defined class of problems it is possible to show some basic properties of a translation of these problems to timed automata models. We will give a generic timed automaton model and show how to obtain a schedule from diagnostic information. A model of the Sidmar steel plant of case study 5 then demonstrates how this approach extends to scheduling problems that do not fit in the class of job shop problems.

The job shop problem will be defined in Section 2.2, timed automata in Section 2.3. Section 2.4 introduces the timed automata model of a job shop problem and we show how to translate traces to schedules. We will then give a description of the Sidmar plant and discuss the timed automaton model in Section 2.5. Section 2.6 presents results for a model of a job shop problem and the Sidmar model. Finally, Section 2.7 discusses which problems arise with this approach, and gives an outlook on the succeeding chapters, that address these problems.

2.2 Job Shop Scheduling

The *job shop scheduling* problem is to find an optimal schedule for set of jobs on a set of machines. Each job is a chain of operations, and machines can only process a limited number of operations at a time. The purpose is to allocate starting times to the operations, such that the maximal completion time is minimal. The job shop problem is known to be NP-hard [GJ79, p. 242]. Many solutions methods like simulated annealing or tabu search [AvLLU94], shifting bottleneck algorithms [AC91], and even hybrid methods [JM99] that combine different approaches have been proposed.

Definition 2.1 *A job shop $\mathcal{P}$ is a tuple $(\mathcal{J}, \mathcal{O}, \mathcal{M}, \mathrm{j}, \mathrm{d}, \mathrm{m}, \mathrm{c}, \prec)$ where*

$\mathcal{J}$ *is a finite set of jobs,*

$\mathcal{O}$ *is a finite set of operations,*

$\mathcal{M}$ *is a finite set of machines,*

$\mathrm{j} : \mathcal{O} \to \mathcal{J}$ *gives for each operation the job it belongs to,*

$\mathrm{d} : \mathcal{O} \to \mathbb{N}\backslash 0$ *defines the duration of each operation,*

$\mathrm{m} : \mathcal{O} \to \mathcal{M}$ *gives the machine on which the operation has to be performed, and finally*

$\prec$ *is a partial order on the set of operations that satisfies*

$$\forall o, p \in \mathcal{O}, o \neq p.\ (o \prec p \vee p \prec o) \Leftrightarrow j(o) = j(p) \tag{2.1}$$

Equivalence (2.1) states that all operations of the same job are totally ordered, and that there is no precedence between operations of different jobs. Similar to the definitions of job shops in [AC91, CP89, AvLLU94] we require that the maximal capacity of machines is one. The definitions of the general job shop in [Vae95] and [DSW98] cover a larger class of problems. They allow machines that can process more than one operation at the same time, and do not require (2.1) to hold.

Definition 2.2 *Let $\mathcal{P}$ be a job shop. A schedule of $\mathcal{P}$ is a function $\mathcal{S} : \mathcal{O} \to \mathbb{N}$ that defines the starting time of each operation. The completion time of an operation is then $S(o) + d(o)$. A schedule $\mathcal{S}$ is feasible if it satisfies, for all $o, p \in \mathcal{O}$:*

$$o \prec p \Rightarrow \mathcal{S}(o) + \mathrm{d}(o) \leq \mathcal{S}(p) \tag{2.2}$$

$$o, p \in \mathcal{O}, o \neq p, \mathrm{m}(o) = \mathrm{m}(p) \Rightarrow \mathcal{S}(o) + \mathrm{d}(o) \leq \mathcal{S}(p) \vee \mathcal{S}(p) + \mathrm{d}(p) \leq \mathcal{S}(o) \tag{2.3}$$

We write $\mathcal{F}(\mathcal{P})$ for the set of all feasible schedules.

Thus, a schedule is feasible if it respects the order between operations of the same job (2.2), and if two operations that use the same machine are not processed simultaneously (2.3). The makespan of a schedule $\mathcal{S}$ is the maximum of the completion times.

Definition 2.3 *Let $\mathcal{P}$ be a job shop. The job shop problem is to find a feasible schedule $\mathcal{S} \in \mathcal{F}(\mathcal{P})$ such that*

$$\max_{o \in \mathcal{O}}(\mathcal{S}(o) + \mathrm{d}(o)) = \min_{\mathcal{S}' \in \mathcal{F}(\mathcal{P})} \max_{o \in \mathcal{O}}(\mathcal{S}'(o) + \mathrm{d}(o)) \tag{2.4}$$

i.e. $\mathcal{S}$ minimizes the maximum of the completion times. We will say that $\mathcal{S}$ is a schedule with minimal makespan.

2.3 Timed Automata

Since we use the model checking tool UPPAAL, we work with networks of timed automata as defined in [BJLY98, LPY97] to model the job shop problem. A timed automaton is a finite automaton over a set of labels *Act*, equipped with a finite set of clocks C, whose values increase uniformly with time. Labels are either local or synchronizing. If label a is synchronizing it has a complement $\bar{a}$, which in turn has as complement $\bar{\bar{a}} = a$. A clock constraint g is a conjunction of atomic constraints of the form $x \sim n$ and $x - y \sim n$ for $x, y \in C$, $n \in \mathbb{N}$ and $\sim \in \{<, \leq, =, \geq, >\}$. We denote the set of all clock constraints with $\mathcal{B}(C)$.

Definition 2.4 (Timed Automaton) *A Timed Automaton over clocks C and actions Act is a tuple (Loc, l_0, E, Inv) where Loc is set of locations, l_0 is the initial location, $E \subseteq Loc \times \mathcal{B}(C) \times Act \times \mathcal{P}(C) \times Loc$ is the set of edges, and $Inv : Loc \rightarrow \mathcal{B}(C)$ assigns invariants to locations. In the case of $(l, g, a, r, l') \in E$, we write $l \xrightarrow{g,a,r} l'$.*

A *Network of Timed Automata* is defined as parallel composition $A_1| \ldots |A_n$ of the timed automata $A_1, \ldots, A_n$ over clocks C and labels *Act*. Let l be the control vector $(l_1, \ldots, l_n)$, with $l_i \in Loc_i$. The control vector where the i-th element l_i is replaced by l'_i will be denoted by $l[l'_i/l_i]$.

Definition 2.5 (Parallel Composition) *The parallel composition $A_1| \ldots |A_n$, with $A_i = (Loc_i, l_{0_i}, E_i, Inv_i)$ is defined to be the timed automata (Loc, l_0, E, Inv) with $Loc = L_1 \times \ldots L_n$, $l_0 = (l_{0_1}, \ldots, l_{0_1})$ and $Inv(l) = Inv_1(l_1) \wedge \ldots Inv_n(l_n)$. The timed automaton A has*

- *a local transition $l \xrightarrow{g,a,r} l'$ iff there exist transition $l_i \xrightarrow{g,a,r} l'_i$ with local label a of automaton A_i such that $l' = l[l'_i/l_i]$*
- *a synchronizing transition $l \xrightarrow{g,a,r} l'$ iff there exists a transition $l_i \xrightarrow{g_i,a,r_i} l'_i$ with synchronizing label a of automaton A_i and a transition $l_j \xrightarrow{g_j,\bar{a},r_j} l'_j$ of automaton A_j such that $g = g_i \wedge g_j$, $r = r_i \cup r_j$ and $l' = l[l'_i/l_i][l'_j/l_j]$.*

The *state* of a timed automaton is a pair (l, v), where $l \in Loc$ and $v : C \rightarrow \mathbb{R}_{\geq 0}$ a valuation of the clocks. We denote by $\mathbb{R}^C$ the set of clock valuations for C. We use $v \in g$ to denote that the clock valuation v satisfies the clock constraint $g \in \mathcal{B}(C)$. Note that the clock valuations that satisfy $g \in \mathcal{B}(C)$ form a convex set. We define the operation $v' = [r \mapsto 0]v$ to be the assignment such that $v'(x) = 0$ if $x \in r$ and $v(x)$ otherwise, and the operation $v' = v + d$ to be the assignment such that $v'(x) = v(x) + d$.

Definition 2.6 (Semantics) *The semantics of a timed automaton A is defined as a labeled transition system with the state-space $L \times \mathbb{R}^C$ with initial state (l_0, v_0) (where v_0 assigns zero to all clocks in C), $v_0 \in Inv(l_0)$ and with the following transition relation:*

- $(l, v) \xrightarrow{\epsilon(d)} (l, v + d)$ *if* $\forall 0 \leq e \leq d : v + e \in Inv(l)$,
- $(l, v) \xrightarrow{a} (l', v')$ *if there exists* g, r *s.t.* $l \xrightarrow{g,a,r} l'$, $v \in g$, $v' = v[r \mapsto 0]$, $v' \in Inv(l')$.

The transitions are decorated with a delay-quantity or an action. A sequence of delays and transitions $(l_0, v_0) \rightarrow (l_1, v_1) \rightarrow (l_2, v_2) \ldots$ with initial state (l_0, v_0) is called *execution*. State (l, v) is called *reachable* if there exists a finite execution with final state (l, v). We denote the final state of an execution α with $\alpha.(l, v)$. We write $exec(A)$ for the set of (finite) executions of a timed automaton A. For $\alpha, \beta \in exec(A)$ we use the notation $\alpha \sqsubset \beta$ if α is a prefix of β. The *span* of a finite execution is defined as the finite sum $\sum_i d_i$ of the delays $\epsilon(d_i)$. By definition, we have if $\alpha \sqsubset \alpha'$ then $span(\alpha) \leq span(\alpha')$.

A forward reachability algorithm searches for a state by enumerating all reachable states. Starting from the initial configuration reachable states are computed until either the state has been found or a fixpoint has been reached. If the model checker finds such a state it can generates diagnostic information that shows how to reach that state. Backwards reachability algorithms in contrast start with a state and computes its predecessors. If an initial set is among those, the state is proven to be reachable.

The semantics of definition 2.6 yield a transition system with uncountable many transitions, and is thus unsuitable for a reachability algorithm. Model checkers like UPPAAL overcome this problem by symbolic semantics based on *symbolic states* of the from $(l, \mathbf{Z})$, with $l \in Loc$ and $\mathbf{Z} \in \mathcal{B}(C)$. The set Z is called *zone*. The symbolic semantics yield a finite transition system, which is appropriate for automatic verification [LPY97]. The next chapter will pay more attention to symbolic state-space exploration.

2.4 From Trace to Schedule

Let $\mathcal{P} = (\mathcal{J}, \mathcal{O}, \mathcal{M}, \mathrm{j}, \mathrm{d}, \mathrm{m}, \mathrm{c}, \prec)$ be a job shop. We model $\mathcal{P}$ as a network of timed automata. For each job $\mathbf{J} \in \mathcal{J}$ we include a timed automaton $TA(\mathbf{J})$, which will be constructed as follows. Let $\mathbf{j}$ be the number of operations $o \in \mathcal{O}$ with $\mathrm{j}(o) = \mathbf{J}$. Let $\mathrm{n}(o)$ be the number of operations of that precede o, i.e. $\#\{p \in \mathcal{O} | p \prec o\}$. Then we will construct automaton $TA(\mathbf{J})$ with control locations $s_0, \ldots, s_{\mathbf{j}}$ and $t_0, \ldots, t_{\mathbf{j}-1}$.

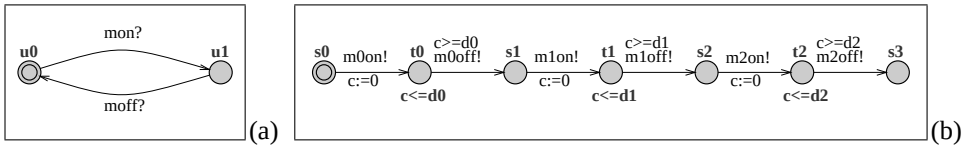


Figure 2.1: Figure (a) shows a timed automaton modeling a machine with capacity 1. Figure (b) shows how to model a job with three operations, and durations `d0`, `d1`, `d2`.

The automaton will be in location $s_{\mathrm{n}(o)}$ before operation o, and in $s_{\mathrm{n}(o)+1}$ after operation o is processed. While operation o is processed automaton $TA(\mathbf{J})$ will be in location $t_{\mathrm{n}(o)}$. To time the duration of the operations we include clock $clock_{\mathbf{J}}$.

We have for each machine two labels in set *Act*. One to models the beginning of an operation on that machine and one to model the end of an operation. For convenience we define two injective mappings $\mathrm{on} : \mathcal{M} \rightarrow Act$ and $\mathrm{off} : \mathcal{M} \rightarrow Act$ with $\mathrm{on}(\mathcal{M}) \cap \mathrm{off}(\mathcal{M}) = \emptyset$. We then have for each operation $o \in \mathcal{O}$ the following transitions:

$$\begin{array}{lcl} s_{\mathrm{n}(o)} & \xrightarrow{\mathrm{on}(\mathrm{m}(o)),\{clock_{\mathbf{J}}\}} & t_{\mathrm{n}(o)} \\ t_{\mathrm{n}(o)} & \xrightarrow{clock_{\mathbf{J}} \geq \mathrm{d}(o_i),\mathrm{off}(\mathrm{m}(o))} & s_{\mathrm{n}(o)+1} \end{array} \tag{2.5}$$

The automaton J is initially in location s_0. Furthermore, we assume that the invariant $clock_{\mathbf{J}} \leq \mathrm{d}(o)$ holds in location $t_{\mathrm{n}(o)}$. [1]

For each machine $\mathbf{M} \in \mathcal{M}$ we include a small timed automaton $TA(\mathbf{M})$.The timed automaton M has only the two following transitions:

$$\begin{array}{lcl} u_0 & \xrightarrow{\overline{\mathrm{on}(\mathbf{M})}} & u_1 \\ u_1 & \xrightarrow{\overline{\mathrm{off}(\mathbf{M})}} & u_0 \end{array} \tag{2.6}$$

The job shop $\mathcal{P}$ with n jobs and m machines is modeled by the composition automaton $P = TA(\mathbf{J_1})| \ldots |TA(\mathbf{J_n})|TA(\mathbf{M_1})| \ldots |TA(\mathbf{M_m})$. We assume in the remainder for simplicity that each job consist of m operations. The job $\mathbf{J}_i$ has thus completed its operations, if automaton $TA(\mathbf{J_i})$ is in control location s_m.

Figure 2.1 illustrates this model for a job shop problem with 3 machines. Table 2.2 shows the complete textual model of the Muth and Thomson problem *mt06*, a job shop problem with 6 machines and 6 jobs, as network of timed automata in UPPAAL. The model contains two templates, one for machines and another for jobs. The template for the machine has just two locations `u0` and `u1`, and two transitions labeled `mon?` and `moff?`. The second template `job` is used to define

[1] This invariant is strictly speaking not necessary, but its use reduces the size of the state-space. See also page 83.

```
chan on0,on1,on2,on3,on4,on5,off0,off1,off2,off3,off4,off5;
process machine(chan mon; chan moff){
state u0, u1;
init u0;
trans u0 -> u1 {sync mon?; },
      u1 -> u0 {sync moff?; };
}
process job(chan m0on,m0off;const d0;chan m1on,m1off;const d1;
chan m2on,m2off;const d2;chan m3on,m3off;const d3;
chan m4on,m4off;const d4;chan m5on,m5off;const d5){
clock c;
state s0,s1,s2,s3,s4,s5,s6,
      t0{c<=d0},t1{c<=d1},t2{c<=d2},t3{c<=d3},t4{c<=d4},t5{c<=d4};
init s0;
trans s0 -> t0 {sync m0on!; assign c:=0; },
      s1 -> t1 {sync m1on!; assign c:=0; },
      s2 -> t2 {sync m2on!; assign c:=0; },
      s3 -> t3 {sync m3on!; assign c:=0; },
      s4 -> t4 {sync m4on!; assign c:=0; },
      s5 -> t5 {sync m5on!; assign c:=0; },
      t0 -> s1 {guard c>=d0; sync m0off!; },
      t1 -> s2 {guard c>=d1; sync m1off!; },
      t2 -> s3 {guard c>=d2; sync m2off!; },
      t3 -> s4 {guard c>=d3; sync m3off!; },
      t4 -> s5 {guard c>=d4; sync m4off!; },
      t5 -> s6 {guard c>=d4; sync m5off!; };
}
M1:=machine(on0,off0);
M2:=machine(on1,off1);
M3:=machine(on2,off2);
M4:=machine(on3,off3);
M5:=machine(on4,off4);
M6:=machine(on5,off5);
//
//  6 BY 6 TEST CASE FROM MUTH AND THOMPSON
//  6  6
//  2  1   0  3   1  6   3  7   5  3   4  6
//  1  8   2  5   4 10   5 10   0 10   3  4
//  2  5   3  4   5  8   0  9   1  1   4  7
//  1  5   0  5   2  5   3  3   4  8   5  9
//  2  9   1  3   4  5   5  4   0  3   3  1
//  1  3   3  3   5  9   0 10   4  4   2  1
//
J1:=job(on2,off2,1,on0,off0,3,on1,off1, 6,on3,off3, 7,on5,off5, 3,on4,off4,6);
J2:=job(on1,off1,8,on2,off2,5,on4,off4,10,on5,off5,10,on0,off0,10,on3,off3,4);
J3:=job(on2,off2,5,on3,off3,4,on5,off5, 8,on0,off0, 9,on1,off1, 1,on4,off4,7);
J4:=job(on1,off1,5,on0,off0,5,on2,off2, 5,on3,off3, 3,on4,off4, 8,on5,off5,9);
J5:=job(on2,off2,9,on1,off1,3,on4,off4, 5,on5,off5, 4,on0,off0, 3,on3,off3,1);
J6:=job(on1,off1,3,on3,off3,3,on5,off5, 9,on0,off0,10,on4,off4, 4,on2,off2,1);
system J0,J1,J2,J3,J4,J5,M0,M1,M2,M3,M4,M5;
```

Table 2.2: UPPAAL model of the Fisher and Thompson problem *mt06* as `xta`-file.

the jobs, and has six on and six off-transitions. The templates are then instantiated according to the description of the job shop problem.

A job shop problem with n job and m machines is usually represented by a n by $2 \cdot m$ matrix, as shown in Table 2.2.[2] Each row of the matrix, which consists of m pairs, defines a job. The first element of each pair defines the machine and the second the duration of the operation. The operations have to be performed in the

[2] The `mt06` problem and other instances of the job shop problem can be found at the url: `ftp://ftp.caam.rice.edu/pub/people/applegate/jobshop/.`

order, in which they appear in the row that defines the job.

Given the composition automaton, we can use UPPAAL to find a state in which all jobs are completed. Since we use UPPAAL, this information will be an execution. Hence, the goal is to obtain an execution α that ends in a state $\alpha.(l.v)$ that satisfies $l_i = s_m, \forall i \in 1, \ldots, n$. We abbreviate this property with Ω. The schedule is obtained by associating an on-transition of an operation o with a starting time $\mathcal{S}(o)$. Execution $\alpha \in exec(P)$ ends with a on-transition of operation o, if

$$\alpha = \alpha' \xrightarrow{\mathrm{on(m}(o))} (l', v') \qquad \wedge \qquad l' = \alpha'.l[J.t_{\mathrm{n}(o)}/J.s_{\mathrm{n}(o)}][M.u_1/M.u_0] \quad (2.7)$$

with $J = TA(\mathrm{j}(o))$ and $M = TA(\mathrm{m}(o))$. This means that as well the label as the change of the control location agree with the operation. We abbreviate property (2.7) with $\Omega_{\mathrm{on}}(o)$. Similarly we can define $\Omega_{\mathrm{off}}(o)$ for executions that end with a off-transition of operation o.

Definition 2.7 *Let $\mathcal{P}$ be a job shop problem and P the corresponding composition automaton. Let $\alpha \in exec(P)$, with $\alpha \models \Omega$. We will say that $\mathcal{S} : \mathcal{O} \to \mathbb{N}$* corresponds *to α, denoted as $\alpha \mapsto \mathcal{S}$, if for all $o \in \mathcal{O}$ there exists a $\alpha' \sqsubseteq \alpha$ such that*

$$\alpha' \models \Omega_{\mathrm{on}}(o) \ \wedge \ \mathcal{S}(o) = span(\alpha') \quad (2.8)$$

A model of the job shop problem is sound and complete, if we can find for each execution that satisfies Ω a corresponding feasible schedule, and if we have for each feasible schedule a corresponding execution that satisfies Ω.

Lemma 2.8 (Soundness) *Let $\mathcal{P}$ be a job shop and P the corresponding timed automaton. Let $\alpha \in exec(P)$ with $\alpha \models \Omega$, and $\mathcal{S} : \mathcal{O} \to \mathbb{N}$ such that $\alpha \mapsto \mathcal{S}$. Then $\mathcal{S} \in \mathcal{F}(\mathcal{P})$.*

Proof It has to be shown that the schedule that assigns to each operation the accumulated delay of the corresponding prefix is feasible. Let $o, p \in \mathcal{O}$ with $o \prec p$, then $\mathrm{j}(o) = \mathrm{j}(p)$ by (2.1). By construction of the automaton $TA(\mathrm{j}(0))$ the off-transition of o has to be taken before the on-transition of p. The guard on the off-transition of o ensures that between the on-transitions of o and p at least $\mathrm{d}(o)$ time units elapsed. This ensures (2.2).

The automata that model the machines guarantee that on and off-transitions of the same machine alternate. Let $o, p \in \mathcal{O}$, with $\mathrm{m}(o) = \mathrm{m}(p)$. We then have that either the off-transition of o precedes the on-transition of p, or that the off-transition of p precedes the on-transition of o. The guards on the transitions then ensure either a delay of $\mathrm{d}(o)$ between the on-transition of o and p, or a delay of $\mathrm{d}(p)$ between the on-transition of p and o. Therefore (2.3) holds. Hence, the schedule that corresponds to α is feasible. □

Lemma 2.9 (Completeness) *Let $\mathcal{P}$ be a job shop and P the corresponding timed automaton. For all feasible schedules $\mathcal{S} \in \mathcal{F}(\mathcal{P})$ there exist an execution $\alpha \in exec(P)$ with $\alpha \mapsto \mathcal{S}$.*

Proof Let $\mathcal{S}$ be a feasible schedule, and let $o_1, \ldots, o_{n \cdot m}$ be the operations ordered with respect to ascending $\mathcal{S}(o_i)$. We want show that we can take for each operation o_i a on-transition at time $\mathcal{S}(o_i)$. First observe, that if the composition automaton is in a state in which some machines are in location `u1`, it is possible to take the corresponding off-transitions, interleaved with the necessary delays, in the order of ascending completion times. Suppose, that we have constructed an execution α_i that ends in an on-transition of o_i. We can then extend this execution, by taking first the off-transitions of operations p with completion times $\mathcal{S}(p) + \mathrm{d}(p) \leq \mathcal{S}(o_{i+1})$, interleaved with delays.

Equation (2.2) then guarantees, that all operations preceding o_{i+1} have been completed, and (2.3) ensures that machine $\mathrm{m}(o_{i+1})$ has been released. We can then take the on-operation of o_{i+1}, preceded by the necessary delay (this might be zero). After taking the last on-transition of $o_{n \cdot m}$, we can take the remaining of off-transitions, and end with a execution that satisfies Ω and has as corresponding schedule $\mathcal{S}$. □

In the timed automaton model we have for each feasible schedule an execution to a final state and vice versa. But, the relation from schedules to executions is not one-to-one, but one-to-many. If the starting or completion times of two operation coincide, then any interleaving of the corresponding transitions will lead to the same schedule. Only if the completion time of the first operation coincides with the starting time of a second, and both operations belong either to the same job or require the same machine, then the first operation has to be completed before the second starts. This yields a huge number of intermediate symbolic states that lead to the same solution, and increases thus the state-space. This is known as partial order problem, and is common to all automata models. The job shop models suffer in particular from this problem, as they incorporate a lot of components that act independently. There are approaches for partial order reduction of timed systems [BJLY98, Min99] that address this problem, but unfortunately they are not yet applicable in a model checker for the full class of timed automata.

The disjunctive graph representation, the most prevalent modeling formalism for job shop problems, avoids these problems. The nodes in the graph represent operations. Two consecutive operations of the same job are connected by a directed arc. Operations that require the same machine are connected by undirected edges. Each orientation of the edges that result in a acyclic graph gives a partial order on the operations. This partial order can then be used to build a feasible schedule. Orienting an edge means that the conflict between two operations is resolved, and

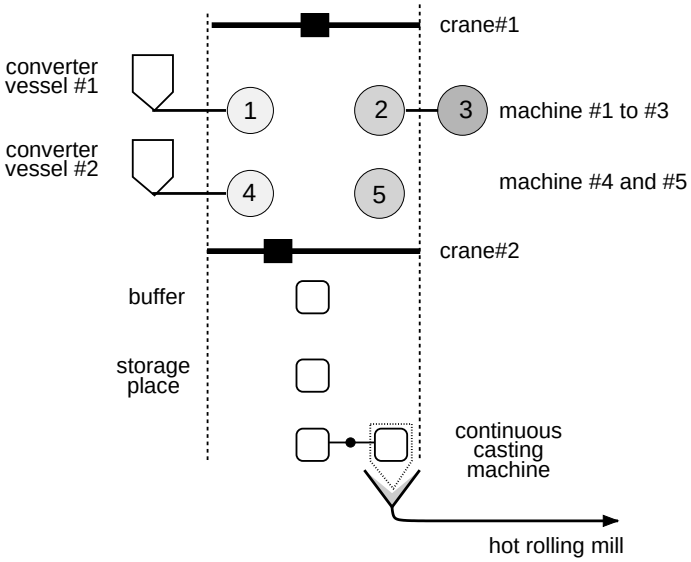


Figure 2.3: Layout of the Sidmar steel plant

by construction only conflicts that might matter are considered.

The disjunctive graph representation allows to branch on the orientation of the edges. Branch and bound algorithms that use this branching scheme have been applied successfully to job shop problems [AC91, CP89]. Local search algorithm can employ this representation, too, for it is possible to define a neighborhood function on disjunctive graphs [AvLLU94]. This algorithm are able to find close to optimal solutions for instances of the job shop problems that are beyond reach for enumerative methods.

All approaches that are based on a disjunctive graph representation, have to build a feasible solution from the graph. It is then sufficient to consider only *left justified* schedules, i.e. no operation can be completed earlier, without either changing the order of the operations on any machine or delaying another operation [Tri90]. It has been proven that there exist always a left justified solution to a job shop problem. The timed automaton approach allows in contrast to obtain the schedule almost directly from the diagnostic information. We showed in this section that this approach can be use for modeling job shop problems. In the next sections we will show that this approach extends also to an industrial case study.

2.5 The Sidmar Steel Plant

2.5.1 Plant description

Plant Layout Case Study 5 of the VHS project considers the part of the Sidmar steel plant in between the converters and the hot rolling mill. This part of the plant consists of two converter vessels, five machines, one storage place, one buffering place, one continuous casting machine, and two overhead cranes on one track. Figure 2.3 depicts the layout of the plant.

The raw pig iron enters the system at one of the converter vessels, where it is poured portion-wise in steel ladles. The pig iron undergoes different treatments in the different machines, depending on desired steel quality. Converter vessel #1 and machine #1 share one car that carries the ladle. This means that there can be at most one ladle in either of these positions. Similarly, converter vessel #2 and machine #4, and machine #2 and machine #3 share a car. The overhead cranes connect the different parts of the plant. Whenever a ladle has to be moved for example from machine #3 to machine #4, an empty crane has to be available.

The steel leaves the system at the continuous casting machine. The casting machine consists of two parts, a holding place and the casting machine itself and works like a merry-go-round. The steel is pored into the hot rolling mill, and the empty ladle is then transported to the storage place by an overhead crane.

Recipes The steel quality depends, as mentioned before, on the order of the different treatments. Machine #1 and machine #4 are identical and so are machine #2 and machine #5. Hence, we have three types of machines that can perform three different types of treatments. A treatment is defined by the type of the machine and a duration. There is an upper bound on the time a load can stay in the system, to prevent it from cooling down to much. This upper bound depends on the quality of the steel. Most recipes consist of at most four treatments. All recipes start with a treatment on machine #1 or machine #4. Next, all recipes require a treatment on machine #2 or machine #5. Some recipes also require a treatment on machine #3 and maybe a final treatment on machine #2 or machine #5.

Timing constraints Except for the timing constraints which are imposed by the recipes there are three other types of timing constraints. First, whenever a ladle is filled at the converter vessel it needs some time before the next load pig iron can be tapped at this vessel. Second, the cranes need some time to pick up and drop a ladle, and to move from one position to another. Finally, a ladle has to wait for a certain amount of time in the holding place before it is allowed to enter the casting machine. After being casted the ladle has to wait for the same amount of time in

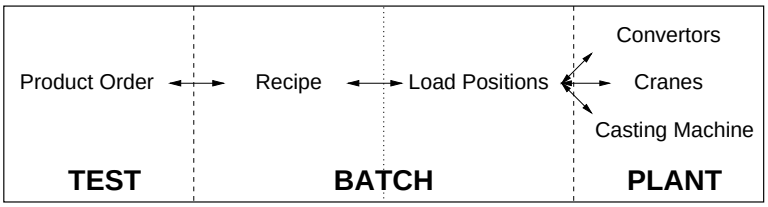


Figure 2.4: Structure of the Sidmar model. Components that synchronize are connected by an arc.

the holding place. The duration of casting a ladle is controllable within known bounds.

Other constraints Since each car can hold at most one ladle, there is no possibility for ladles to pass each other. A ladle cannot move for example from the converter# 1 to machine #2, if another ladle is at machine #1. The cranes cannot pass each other, and only one crane can reach the bottom section where the ladles enter the continuous casting machine. The buffer between the second track and the storage place can hold at most five ladles. It can be used to pass a ladle from one crane to the other.

A ladle can only leave the casting machine if there is already a filled ladle at the holding place (except for the case it is the last ladle). As soon as the pouring out of a steel ladle has been completed, this ladle is removed and casting of a new ladle starts. The casting machine works like a merry-go-round. It places the filled ladle in the casting machine while it removes the empty ladle and places it back into the holding place. This guarantees that the casting machine is continuously in use.

Objective The foregoing constraints define the physical and technical requirements of the plant. To meet the economic constraints we want that the different steel qualities enter the continuous casting machine in a predefined order.

2.5.2 The Timed Automata Model

We will use UPPAAL's input language for modeling the Sidmar steel plant. Systems in UPPAAL are modeled as networks of timed automata[AD94]. The different components of the system are modeled as timed automata. The automata are combined using UPPAAL's operation of parallel composition with binary (handshake) synchronization. Since in our model more than two components need to synchronize, we will use UPPAAL's concept of committed locations. Whenever a com-

mitted location is entered no delay is allowed and the next action transition must be an outgoing transition of that location. We use binary arrays to ensure mutual exclusion. A process is only allowed to enter a certain location if a corresponding bit is zero. It then sets the bit to one and releases it as soon as it leaves the location. For a detailed introduction to UPPAAL see [LPY97].

Overview An overview of the structure of the model is given in Figure 2.4. The model distinguishes between the batches and the plant. The state of a batch is defined by the position of the load in the plant and the position within the recipe and the time that has passed since the load entered the system. A timed automaton that keeps track of the load is included into the model. This TA resembles the layout of the plant. Secondly, the model includes a linear (non branching) automaton for the recipe that determines the duration and order of the treatments. Location invariants in this automaton guarantee that the load cannot spend more time in the system than specified.

In parallel with the TAs who model the batches and the plant we have a test automaton. This is a linear automaton that reaches control location `final` only, if the ladles enter the casting machine in the prescribed order. The recipes synchronize with the test automaton before the ladle enters the bottom section of the plant. The recipes synchronize with the load automaton, which encode the positions of the ladle, to ensure that a treatment is performed by the proper machine. Figure 2.4 shows which TAs synchronize with each other.

The model allows to translate the question whether a feasible schedule exists into the question whether a state is reachable in which all recipes are completed. The diagnostic information then gives the desired schedule, since we assume that all times are either deterministic or controllable, i.e. it they are either time points or they the plant can realize each time point in an interval. The original description [BS99] states that the duration of a treatment is only known within certain bounds. We take the upper bound as duration of the treatment, since each machine can also function as short term buffer.

The timed automaton model in this chapter differs in some points from the model in [Feh99]. A main difference is that the model as presented in this chapter contains modifications to ease the verification, that do not affect the correctness of the model. Furthermore some ambiguities in the informal description have been clarified, which led to modifications.

Test Automaton Figure 2.5(a) shows an example of an automaton that tests whether the batches arrive at the casting machine in the prescribed order. The automaton in this example can reach the state `final` only, if a batch of quality 2

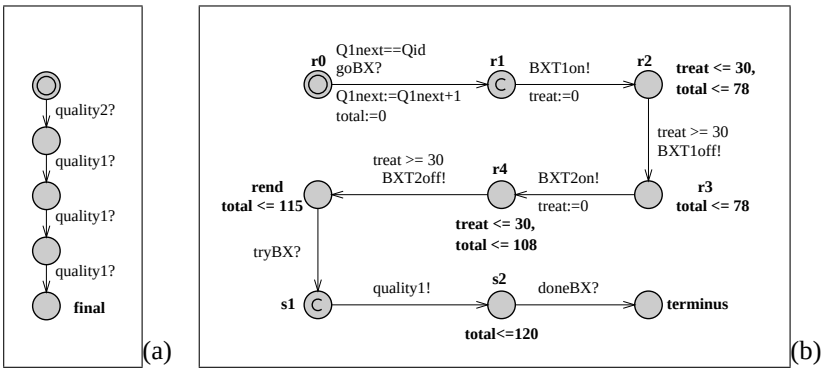


Figure 2.5: (a) The test automaton that guarantees that the prescribed order of qualities. (b) This automaton defines the recipe to produce steel of quality 1. Locations decorated with C are committed

arrives first, succeeded by three loads of quality 1.

Recipe The automaton in Figure 2.5(b) models a recipe to produce a batch of quality 1. As mentioned before, the model includes one automaton for each batch. Each automaton has two local clocks; clock `total` measures the overall time the batch has spend in the system, and clock `treat` times the operations of the recipe.

When batch `X` enters the system, then the recipe will take the transition labeled `goBX?`. The system incorporates a lot of symmetry, mostly since all batches of the same quality are modeled the same. The model checker may explore a lot of states that are syntactically different, but which should be considered equivalent due to symmetry. To reduce the symmetry we number batches of the same quality, using a variable `Q1next`. The guard and the assignment on transition `goBX?` ensure that batches of the same quality start in the order of their increasing numbers. A more systematic approach would be to incorporate symmetry reduction techniques within the tool UPPAAL, as in the untimed model checker Murphy [ID93].

The example recipe in 2.5(b) includes one operation of type 1 and one of type 2, both with a duration of 30 time units. Batch `X` starts a operation of type 2 by taking the transition labeled `BXT1on!` to location `r2`. The transition resets clock `treat` to zero. The invariant `treat<=30` ensures that this operation cannot last longer than the required 30 time units. The transition labeled `BXT1off!`, guarded by `treat>=30`, marks the end of the operation. The subsequent operation of type 2 is modeled similarly. In this example we assume that we have two converter vessels, one to fill empty ladle at machine #1, and one to fill ladles at machine #4. The converter cannot fill a ladle as long as a batch is processed in the neighboring machine, since both share the same car. But since a batch has also to receive its

first treatment at that machine, we can safely combine both operations, and declare location `r1` as committed.

The batch may try to enter the holding place of the casting machine if all necessary operations are completed. This is modeled by the transition labeled `tryBX`. But a batch should only try to enter the holding place if it is of the desired quality. The recipe has therefore to synchronize with the test automaton. The location `s1` is committed to guarantee that both transition are taken in one atomic step. Finally, when the batch `X` enters the casting machine, transition `doneBX` will be taken. The invariants on the locations ensure that the batch arrives at the casting machine before its deadline, which is in this example 120 time units. The invariant in location `rend` is `total<=115`, since has still to wait for at least another 5 time units in the holding place, before it may enter the casting machine. The invariants in the other location are chosen similarly.

Load Automaton The largest automata in the model are the automata that represent the position of the load in the system (Figure 2.6). The model includes for each batch of steel one load automaton. The transitions `fillI` and `fillII` model filling a ladle at one of the two converter vessels. The automaton is in location `machine1` to `machine5` if the corresponding batch is processed in these machines.

The loads are moved by the cranes along locations `c0` to `c4`. Only one crane can reach the bottom section. This allows us to simplify the model for this part as follows. Moving a load from position `c4`, which corresponds to a position next to the storage place, to the holding place can be modeled by a sequence of transitions, tied together with committed locations. Transition `tryBX` ensures that a load enters the holding place only if it has undergone all necessary treatments. Transition `cvdown` models the transport by the crane to the casting machine, and `incast` places the ladle into the holding place.

If the ladle has waited for required time, it may enter the casting machine. Entering the casting machine corresponds to transition `turn?`. The recipe is then completed, which is modeled by transition `doneBX!`. The empty ladle can then leave the casting machine via transition `nrut?`, and move to the storage place via transitions `cvup!`, `outcast!` and `civdown!`. Transition `cvup!` and `civdown!` synchronize with the crane.

The car at machine #2 and machine #3 can be used as a buffer. If a ladle spends some time waiting in that position before it is processed in one of the machines, it might as well being processed first and wait after completion of the treatment. We model this position by two locations `i1a` and `i1b`. Location `i1a` is committed. This modification ensures that a load either waits without being processed, or is

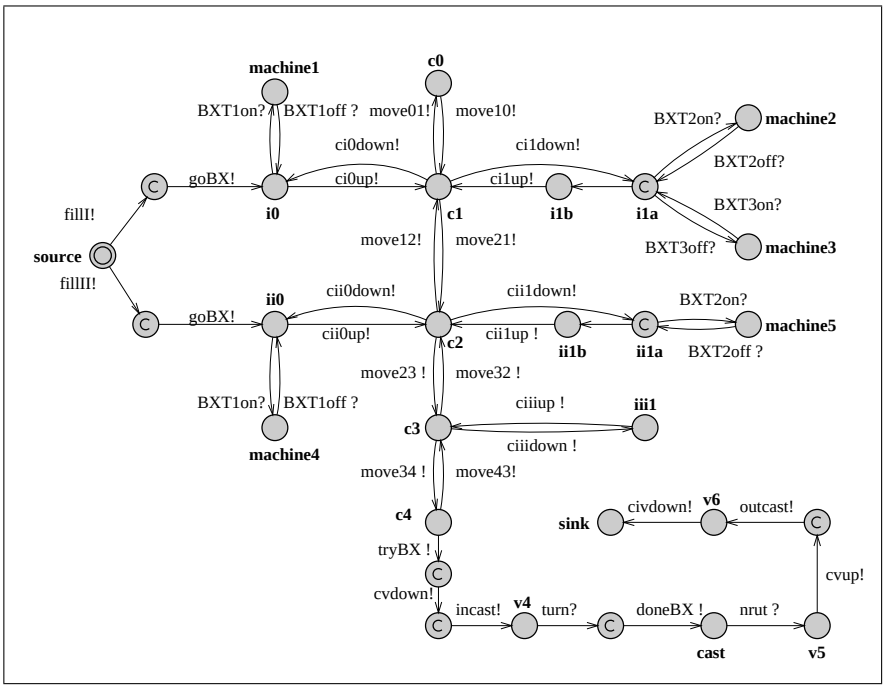


Figure 2.6: This timed automaton keeps track of the position of a load.

processed before it starts waiting. We modeled the car at machine #5 similarly.

Converter Figure 2.7(a) shows the model for converter vessel #1. The converter in this example can fill a ladle with pig iron once every 10 time units. The model uses local clock `t` to establish this constraint. Before it starts filling a ladle one has to guarantee that the car at machine #1 is free. The bit `posI[0]` encodes whether this car is used. The guard `posI[0]==0` ensures that no load is in location `i0`. The automaton can then take transition `fillI` and set `posI[0]` to one.

Casting machine The automaton in Figure 2.7(b) has transitions `incast?` to model a full ladle that enters the holding place, and transitions labeled `outcast?`, that synchronize with a load automaton, to model empty ladles that leave the holding place. An empty ladle can leave the casting machine and enter the holding place only, if there is a full ladle present in the holding place. This constraint establishes a continuous flow of steel to the hot rolling mill.

Transitions labeled `turn!` synchronize with batches that enter the casting machine, and transitions labeled `nrut!` with batches that leave the casting machine.

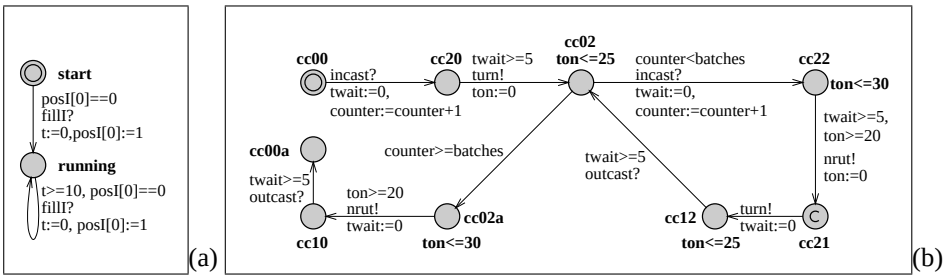


Figure 2.7: (a) Timed automaton for a converter. (b) Timed automaton for the casting machine.

The transitions from location `cc00` to `cc02` model the initialization phase when the first ladle enters the casting machine.

The transitions from `cc02` via `c21` back to `cc02` models be behavior of casting machine up to the last batch. The casting machine repeatedly accepts a full batch, turns and releases an empty batch. Turning the casting machine is modeled in two steps; one to remove the empty ladle from and another to put the other into the casting machine. Since location `cc21` is committed, these steps are taken atomically.

To model the required waiting times and the duration of casting a ladle we use two local clocks `twait` and `ton`. The ladle has to wait for 5 time units when it enters the holding place. Casting a ladle takes between 20 and 30 time units. The duration of casting steel is controllable within these bounds.

The last ladle can leave the casting machine, even if no new ladle is present in the holding place. The transitions from location `cc02` to `cc00a` model this procedure. Integer variable `counter` counts the number of batches that have been completed, and constant `batches` gives the number of `batches` that has to be produced.

Cranes There are two cranes that cannot pass each other. The topmost crane cannot, as mentioned before, reach the bottom section of the plant with the casting machine. The two cranes are therefore modeled differently. Figure 2.8 depicts the model of the top most crane, crane #1. If the crane does not carry a load it is in location `c0bot` to `c3bot`. Each of this locations model a part of the track. Moving the crane from on part to another takes one time unit. Array `cpos` ensures that the cranes cannot be at the same position at the same time[3]. Local clock `tCA` is used to time the movements of the crane.

[3] This array records redundant information on the location of the cranes. It could be omitted if UPPAAL would allow to refer to locations in guards.

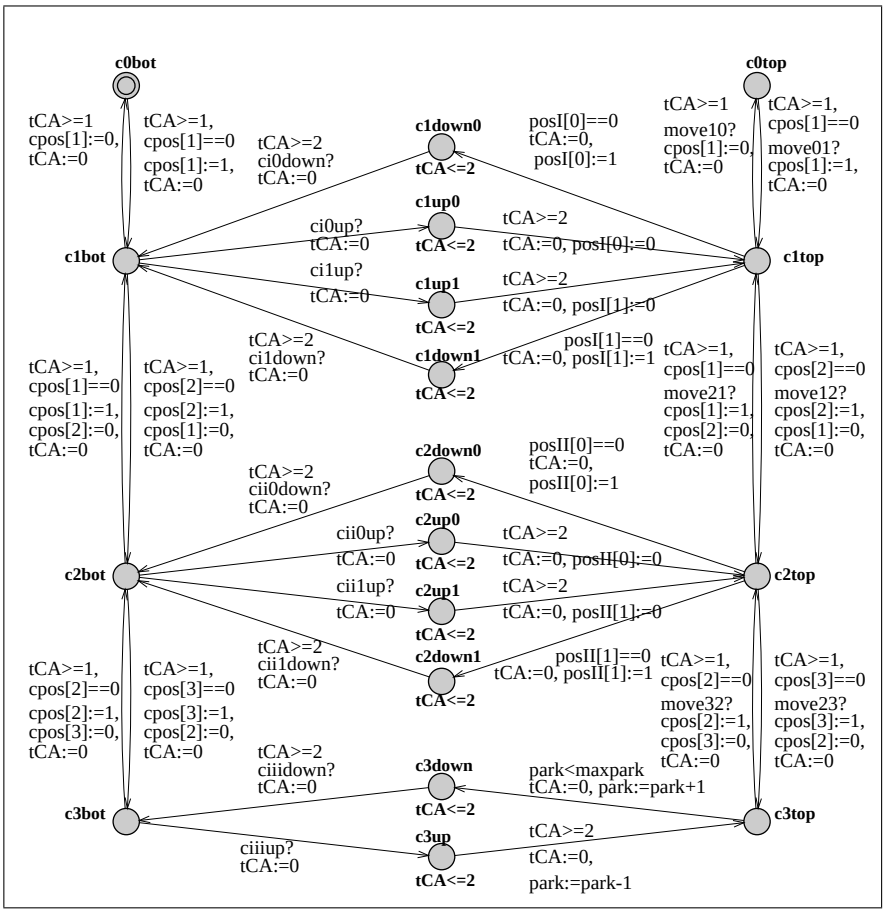


Figure 2.8: Timed automaton model of crane #1

If the crane picks up a load it moves from location `c1bot`, `c2bot` or `c3bot` to location `c1top`, `c2top` or `c3top`, respectively. Picking up a load is modeled as two steps. Firstly, the crane synchronizes with the load. After two time units the crane releases the position of the load, i.e. it sets the corresponding element of array `posI` or `posII` to zero. If it picks up a load from the buffer it decreases the counter of buffered loads `park` by one.

Putting down a load is also modeled as two transitions. The first transition can only be taken if the position is free. It then set the corresponding element of `posI` or `posII` to one. If the crane puts down a load into the buffer it has to make sure that the number of load in the buffer does not exceed `maxpark`. The second transition can be taken after two time units. This transition synchronizes with the

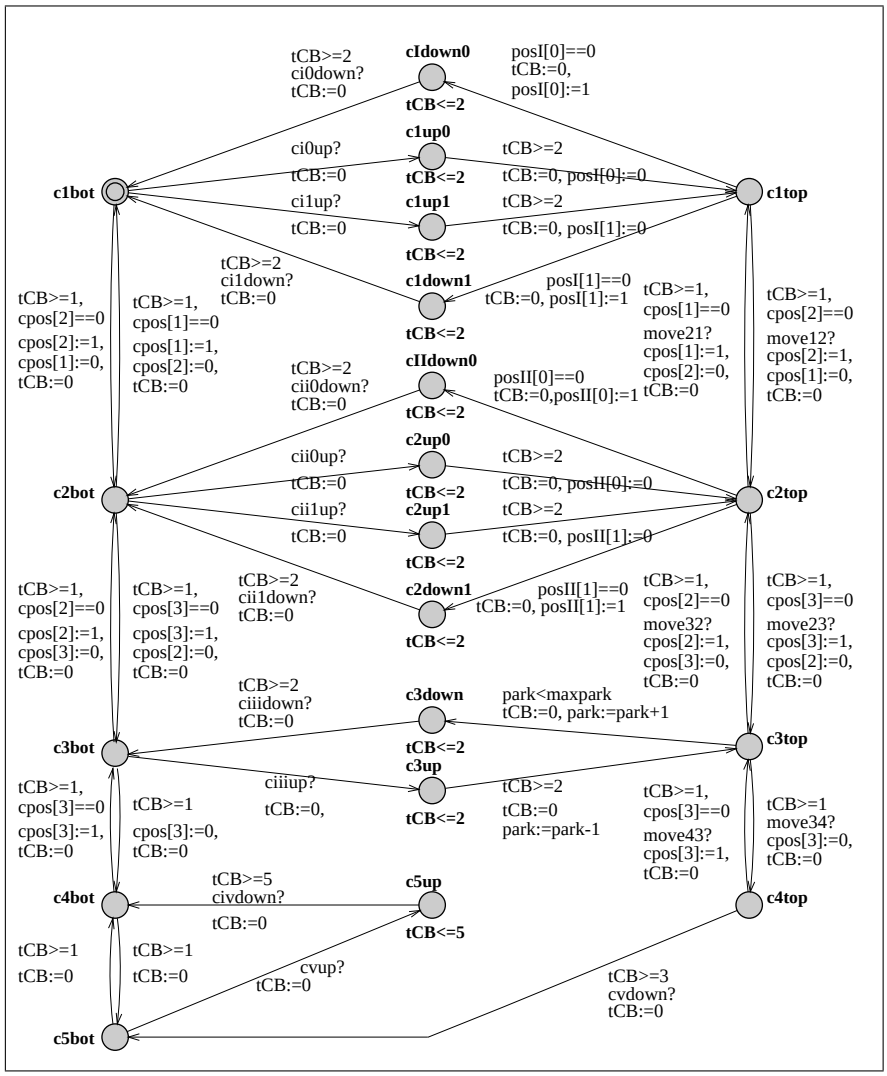


Figure 2.9: Timed automaton model of crane #2

load. The second crane is basically modeled as the first crane. They differ only for parts of the plant only one them can reach. The crane #2 for example cannot reach the top most position of the plant. The model does consequently not include locations `c0bot` and `c0top`. For crane #1 cannot reach the bottom section, we can simplify this part of the model of crane #2. It is for instance not necessary to use the shared array `cpos` to guarantee mutual exclusion. As a consequence we can also combine and omit transitions.

If crane enters the bottom section with a full load it can put it only into the holding place or move back to the top. We can therefore model moving the crane downwards and putting the load down into the holding place as a single operation that takes 3 time units. Similarly, we can combine the operations that pick up an empty ladle from the holding place, move it to the storage place, and put it down there as one operation. Removing a full ladle from the holding place and moving an empty ladle to the top section of the plant are useless and thus not modeled.

2.6 Results

We showed in this chapter how to model a job shop problem and an industrial scheduling problem as networks of timed automata. In this section we will present experimental results for instances of these problems.

The UPPAAL verifier offers two ways to explore the state-space. Depending on whether the list of encountered but unexplored states is a stack or a queue we are searching depth-first or breadth-first, respectively. The breadth-first strategy is not able to find a trace for the *mt06*-problem in Figure 2.2, due to limited memory. The depth-first strategy yields a schedule with completion time 197. This is at the same time the worst possible schedule, since this is the sum of all durations of the *mt06*-problem. This solution is obtained by scheduling first all operations of one job, then all operations of another job, etcetera.

The UPPAAL simulator allows us to apply random search. In this case the simulator selects at random the next transition. We apply this strategy to the *mt06*-problem which has an optimal solution with makespan 55. The model has 12 synchronizing transitions for each job automaton. Since these automata have no cycles it is guaranteed that the 72nd successor of the initial state corresponds to a feasible schedule, no matter how we search for a feasible solution. The generated schedules during 10 experiments with random search had an average completion time of 94.5. The maximal makespan was 120, the minimal makespan was 87. The schedule with the makespan 120 was the only one that took longer than 100 time units.

A different approach to find better feasible schedule uses UPPAALs concept of urgent transitions. We declare all on transitions urgent, i.e. if they are enabled time may not advance. This heuristic will not always lead to an optimal solution, since not only unnecessary delays but also useful delays might be omitted. The verifier found with a depth-first search for this greedy approach a trial schedule with makespan 70.

We combined this approach with some other heuristics that estimated the remaining time and pruned branches if the global clock increased the given upper

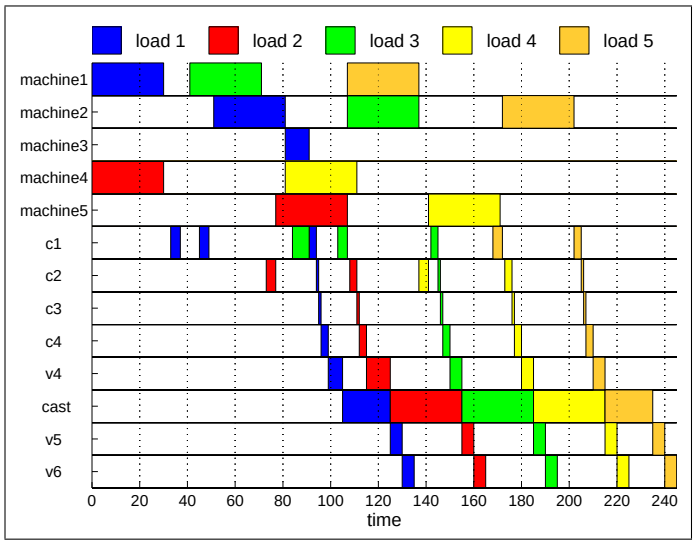


Figure 2.10: Schedule for five batches. This chart depicts which load is in which location at a certain time.

bound minus the remaining time. For each location in a job one can for example determine the duration of all remaining operations of that job. The remaining time has to be bigger that value. We then used the verifier to produce iteratively schedules with a smaller makespan by decreasing the upper bound. We were able to find a schedule with makespan 57, and subsequently that there was no schedule with a makespan smaller than 57. In our attempts to bound the search space by introducing urgency we obviously pruned also the branch that contains the optimal solution with makespan 55.

The model of the Sidmar steel plant contains already some modifications to ease the verification, which were mentioned before. We used depth-first search to find a schedule for this model. We were able to find a schedule for a model with 5 batches, only one crane and no buffer within 2934 seconds cpu time on a Pentium III 500MHz (Figure 2.10). This schedule is not optimal and contains useless delays and transitions; it happens for example that a load gets picked up to be put down in the very same position, immediately afterwards.

Earlier experiments in [Feh99] report schedules for up to three batches for an alternative model. The reason for the different results is not only the different modeling. An important reason is that the results are sensitive to the order in which components appear in the system definition in the textual model. The system description in 2.2, for instance, is the last line that sums up all automata in the composition. UPPAAL evaluates the transitions in the order in which they appear

in the composition automata. Changing the order in the system definition changes also the evaluation order and can thus influence the search order. Experiments with different orders led ultimately, for the model as presented in this chapter, to a schedule for five batches.

2.7 Conclusion and Outlook

This chapter shows that timed automata are suitable for modeling scheduling problems. A model checker can then be used to find a schedule. We can introduce an upper bound on the makespan if we want to improve on this schedule. The language of timed automata allows to naturally model a broad class of scheduling problems that may incorporate a nontrivial combination of deadlines, due dates and other technical constraints. Furthermore, it allows to utilize the efforts that are made to represent and store huge search spaces efficiently

This approach has however some drawbacks. First, it allows only to solve constraints problems; given a deadline they can find a schedule if one exists. In order to get better schedules one has to decrease this deadline. One would prefer to search for an optimal solution automatically. Another drawback is that one has to search either depth or breadth-first. But in many cases they will not find a close to optimal solution or they will find no solution at all. There are only coarse ways to influence the search order, like changing the system description.

These drawbacks derive from the fact that the model checker was not intended to solve scheduling problems. It is tailored towards exploring the full state-space, since one assumes that the properties will eventually be proven to be true for all reachable states. If the algorithm finds an error, it stops; there is no need to continue the search.

It seems therefore promising to adapt techniques from scheduling algorithms, to tackle the problems of the model checking approach to scheduling. A natural candidate for this are branch and bound algorithms, that search in a tree structure for feasible solutions with minimal costs. Like the model checking algorithm, they enumerate the possible solutions. What solutions are considered feasible and how to compute the costs depends on the application area. Branch and bound algorithms can be characterized by four basic rules [Mit70, Tri90].

B1 The branching rule defines how to calculate the successors of a node in the search tree.

B2 The selection rule defines which node in the waiting list to branch from next.

B3 The bounding rule defines how to compute a lower bound on the costs of the complete solutions that can be obtained from a partial solution. This lower

bound allows to prune branches with a lower bound larger than the cost of the best solution found so far. A tight lower bound in combination with a good trial solution can bound the search space effectively.

B4 The elimination rule defines how to recognize branches that cannot contain an optimal solution. A partial solution can be eliminated if is dominated by another partial solution, this means it can be proven that the costs of the best completion of the second is less or equal to the costs of the best completion of the first.

Our approach is based on the timed automata model; the nodes in search tree are symbolic states. The next chapter introduces *Linearly Priced Timed Automata*, who extend timed automata with a notion of cost to that goes beyond time optimality. The branching rule B1 is then defined by the symbolic semantics of linearly priced timed automata. Along with the notion of cost we define a notion of a better symbolic state, and thus the elimination rule B4. We introduce in this chapter an algorithm to compute the optimal execution, and show that it terminates. This algorithm uses a rather inefficient data structure to represent sets of states.

Chapter 4 presents a efficient data structure for the sub-class of *Uniformly Priced Timed Automata*, and Chapter 5 an efficient data structures for the full class of *Linearly Priced Timed Automata*. Both chapters use a modified algorithm that allows to define heuristic search orders. It is possible to manually define a selection rule B2. The algorithm allows also to define an estimate on the remaining cost. This allows to derive a tight bounding rule B3 in order to restrict the state-space. Several experiments with prototype implementations indicate that this approach is competitive with other approaches. Finally, Chapter 6 describes how the results on heuristics extend to a more general class of hybrid systems.

3

Minimal-Cost Reachability for Linearly Priced Timed Automata

3.1 Introduction

This chapter introduces the model of *linearly priced timed automata* as an extension of timed automata with *prices* on both transitions and locations: the price of a transition gives the cost for taking it and the price on a location specifies the cost *per time-unit* for staying in that location. This model can capture not only the passage of time, but also the way that e.g. tasks with different prices for use per time unit, contribute to the total cost. For this model we consider the minimum-cost reachability problem: given a linearly priced timed automaton and a target state, determine the minimum cost of executions from the initial state to the target state. This problem generalizes the minimum-time reachability problem for ordinary timed automata. Similar and independent work has been presented by Alur et al. [ATP01]. They reduce the optimization problem to a shortest-path problem in a finite directed graph, and obtained additionally complexity bounds for the optimization problem.

The minimum-time reachability problem together with the controller synthesis problem has been solved in [AM99], using a backward fix-point computation. Niebert et al give in [NTY00] an alternative solution based on forward reachability analysis. The present chapter extends this work and settles an open problem given in [AM99]. It extends also the work in [ACH97] which provides an algorithm for computing the accumulated delay in a timed automata. We prove decidability of the minimum-cost reachability problem by offering an algorithmic solution, which is based on a combination of branch-and-bound techniques and a new notion

This chapter is based on the publication:

[BFH+01a] G. Behrmann, A. Fehnker, T.S. Hune, K.G. Larsen, P. Petterson, J.M.T. Romijn and F.W. Vaandrager. *Minimum-Cost Reachability for Linearly Priced Timed Automata.* HSCC'01.

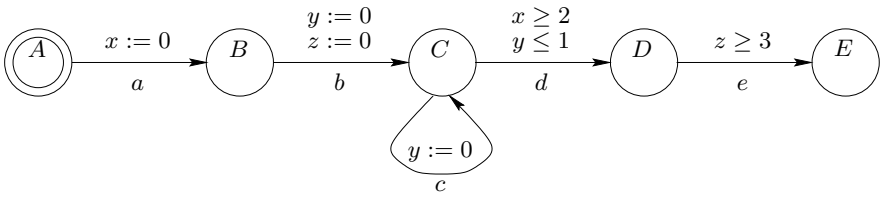


Figure 3.1: Timed automata model of scheduling example.

of priced regions. The latter allows symbolic representation and manipulation of reachable states together with the cost of reaching them.

As an example consider the very simple static scheduling problem represented by the timed automaton in Figure 3.1 from [NTY00], which contains five 'tasks' $\{A, B, C, D, E\}$. All tasks are to be performed precisely once, except task C, which should be performed *at least* once. The order of the tasks is given by the timed automaton, e.g. task B must not commence before task A has finished. In addition, the timed automaton specifies three timing requirements to be satisfied: the delay between the start of the first execution of task C and the start of the execution of E should be at least 3 time units; the delay between the start of the last execution of C and the start of D should be no more than 1 time unit; and, the delay between the start of B and the start of D should be at least 2 time units, each of these requirements are represented by a clock in the model. Using a standard timed model checker we are able to verify that location E of the timed automaton is reachable. This can be demonstrated by a trace leading to that location[1]:

$$(A,0,0,0) \xrightarrow{a} \xrightarrow{\epsilon(1)} (B,1,1,1) \xrightarrow{b} \xrightarrow{\epsilon(1)} (C,2,1,1) \xrightarrow{d} \xrightarrow{\epsilon(2)} (D,4,3,3) \xrightarrow{e} (E,4,3,3) \tag{3.1}$$

The above trace may be viewed as a feasible solution to the original static scheduling problem. Nevertheless, given an optimization problem, one is often not satisfied with an arbitrary feasible solution but one desires solutions which are *optimal* in some sense. When modeling a scheduling problem an obvious notion of optimality is that of minimal makespan, i.e. minimum accumulated time. For the timed automaton of Figure 3.1 the trace of (3.1) has an accumulated time-duration of 4. This, however, is not optimal as witnessed by the following alternative trace, which by exploiting the looping transition on C reaches E within a total of 3 time-units:

$$(A,0,0,0) \xrightarrow{a} \xrightarrow{b} \xrightarrow{\epsilon(2)} (C,2,2,2) \xrightarrow{c} (C,2,0,2) \xrightarrow{d} \xrightarrow{\epsilon(1)} (D,3,1,3) \xrightarrow{e} (E,3,1,3) \tag{3.2}$$

In fact, 3 is the minimum time for reaching E. Figure 3.2 gives a linearly priced extension of the timed automaton from Figure 3.1. Here, the price of location D

[1] Here a quadruple (X, v_x, v_y, v_z) denotes the state of the automaton in which the control location is X and where v_x, v_y and v_z give the values of the three clocks x, y and z, respectively.

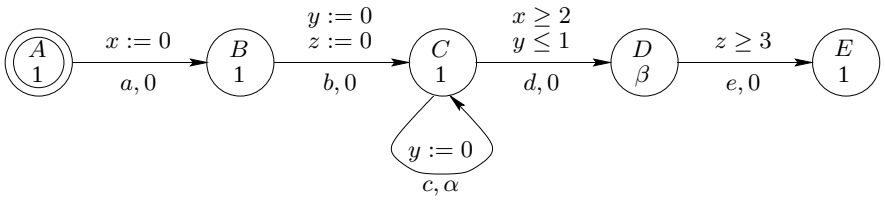


Figure 3.2: Linearly priced timed automata model of scheduling example.

is set to β and the price on all other locations is set to 1 (thus simply accumulating time). The price of the looping transition on C is set to α, whereas all other transitions are free (price 0). Now for $(\alpha, \beta) = (1, 3)$ the costs of the traces (3.1) and (3.2) are 8 and 6, respectively (thus it is cheaper to actually exploit the looping transition). For $(\alpha, \beta) = (2, 2)$ the costs of the two traces are both 6. In this case it is immaterial whether the looping transition is taken or not. In fact, the optimal cost of reaching E will in general be the minimum of $2 + 2 \cdot \beta$ and $3 + \alpha$, and the optimal trace will include the looping transition on C depending on the particular values of α and β.

In contrast to minimum-time reachability for timed automata, the minimum-cost reachability problem for linearly priced timed automata requires the development of new data structures for symbolic representation and the manipulation of reachable *sets* of states *together with* the cost of reaching them. In this chapter we put forward one such data structure, namely a priced extension of the fundamental notion of *clock regions* for timed automata [AD94].

The remainder of the chapter is structured as follows: Section 3.2 formally introduces the model of linearly priced timed automata together with its semantics. Section 3.3 develops the notion of priced clock regions, together with a number of useful operations on these. The priced clock regions are then used in Section 3.4 to give a symbolic semantics capturing the cost of executions. Section 3.5 gives an example on how to compute a symbolic state-space. An algorithm that solves the minimum-cost problem is presented in Section 3.6. Finally, in Section 3.7 we give some concluding remarks.

3.2 Linearly Priced Timed Automata

Linearly priced timed automata extend the model of timed automata [AD94] with prices on both locations and transitions. Dually, linearly priced timed automata may be seen as a special type of linear hybrid automata [Hen96] or multi-rectangular automata [PKHWT98] that represent the accumulation of prices (i.e. the cost) by a

single continuous variable. In contrast to known undecidability results for these classes, minimum-cost reachability is computable for linearly priced timed automata. An intuitive explanation for this is that the additional (cost) variable does not influence the behavior of the automata.

As in Section 2.3, let C be a finite set of clocks, and $\mathcal{B}(C)$ the set of conjunctions of atomic clock constraints of the form $x \sim n$ for $x \in C$, $n \in \mathbb{N}$ and $\sim \in \{<, \leq, =, \geq, >\}$. Note that for each timed automaton that has constraints of the form $x - y \sim n$, there exists a strongly bisimilar timed automaton with only constraints of the form $x \sim n$. Therefore, the results in this chapter are applicable to automata having constraints of the type $x - y \sim n$ as well.

Definition 3.1 (LPTA) *A Linearly Priced Timed Automaton (LPTA) over clocks C and actions Act is a tuple (Loc, l_0, E, Inv, P) where Loc is a finite set of locations, l_0 is the initial location, $E \subseteq Loc \times \mathcal{B}(C) \times Act \times \mathcal{P}(C) \times Loc$ is the set of edges, $Inv : Loc \to \mathcal{B}(C)$ assigns invariants to locations, and $P : (Loc \cup E) \to \mathbb{N}$ assigns prices to both locations and edges. In the case of $(l, g, a, r, l') \in E$, we write $l \xrightarrow{g,a,r} l'$.*

Definition 3.2 (Semantics) *The semantics of a LPTA A is defined as a transition system with the state-space $Loc \times \mathbb{R}^C_{\geq 0}$, with initial state (l_0, v_0) (where v_0 assigns zero to all clocks in C), and with the following transition relation:*

- $(l, u) \xrightarrow{\epsilon(d),p} (l, v + d)$ *if* $\forall 0 \leq e \leq d : v + e \in Inv(l)$, *and* $p = d \cdot P(l)$.
- $(l, v) \xrightarrow{a,p} (l', v')$ *if there exists g, r such that* $l \xrightarrow{g,a,r} l'$, $v \in g$, $v' = v[r \mapsto 0]$, $v' \in Inv(l')$ *and* $p = P((l, g, a, r, l'))$.

Note that the transitions are decorated with two labels: a delay-quantity or an action, together with the cost of the particular transition. For determining the cost, the price of a location gives the cost rate of staying in that location (per time unit), and the price of a transition gives the cost of taking that transition. In the remainder, states and executions of the transition system for LPTA A will be referred to as states and executions of A.

Given LPTA $A = (L, l_0, E, I, P)$, let $\hat{A} = (L, l_0, E, I)$ be the corresponding TA, obtained by dropping the prices. It is trivial to show that by dropping the costs in the executions of A, we obtain exactly the executions of $\hat{A}$.

Definition 3.3 (Cost) *Let $\alpha = (l_0, v_0) \xrightarrow{a_1,p_1} (l_1, v_1) \ldots \xrightarrow{a_n,p_n} (l_n, v_n)$ be a finite execution of LPTA A. The* cost *of α,* cost(α)*, is the sum $\Sigma_{i \in \{1,\ldots,n\}} p_i$. The minimal cost of reaching a given state (l, v) is the infimum of the costs of finite executions*

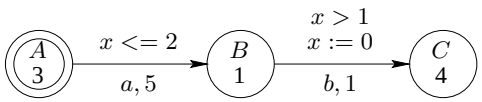


Figure 3.3: An example LPTA.

ending in (l, v). *We denote this infimum with* mincost$((l, u))$. *Similarly, the minimal cost of reaching a location* l *is the infimum of the costs of finite executions ending in a state of the form* (l, u), *thus* mincost(l) *equals* $\inf\{\text{cost}(\alpha)|\alpha$ a run ending in location $l\}$

Example Consider the LPTA of Fig. 3.3. The LPTA has a single clock x, and the locations and transitions are decorated with guards and prices. A sample execution of this LPTA is for instance:

$$(A, 0) \xrightarrow{\epsilon(1.5),4.5} (A, 1.5) \xrightarrow{a,5} (B, 1.5) \xrightarrow{b,1} (C, 1.5)$$

The cost of this execution is 10.5. In fact, there are executions with cost arbitrarily close to the value 7, obtainable by avoiding delaying in location A, and delaying for more than one time unit in location B. Due to the infimum definition of mincost, it follows that mincost$(C) = 7$. Note, however, that because of the strict comparison in the guard of the second transition, no execution actually achieves this cost. □

The need for infimum in the definition of minimum cost executions arises from linearly priced timed automata with strict bounds in the guards, such as the one shown in Figure 3.3. Due to the use of infimum, a linearly priced timed automaton is not always able to realize an execution with the exact minimum cost of the automata, but will be able to realize one with a cost infinitesimally close to the minimum value. If all guards include only non-strict bounds, the minimum cost trace can always be realized by the automaton. This fact can be shown by defining the minimum-cost problem for executions covered by a given symbolic trace as a linear programming problem.

3.3 Priced Clock Regions

For ordinary timed automata, the key to decidability results has been the valuable notion of *region* [AD94]. In particular, regions provide a finite partitioning of the uncountable set of clock valuations, which is also stable with respect to the various operations needed for exploration of the behavior of timed automata (in particular the operations of delay and reset).

In the setting of linearly priced timed automata, we put forward a new extended notion of *priced region*. Besides providing a finite partitioning of the set of clock-valuations (as in the case of ordinary regions), priced regions also associate costs to each individual clock-valuation within the region. As we shall see in the following, priced regions may be presented and manipulated in a symbolic manner and are thus suitable as an algorithmic basis.

We divide a clock valuation v in its integer part $\lfloor v \rfloor$ and its fractional part $\text{frac}(v)$. Let $e_r : C \to \{0,1\}$ for $r \subseteq C$ be the function that assigns 1 if $x \in r$ and 0 otherwise. We then trivially have that $v = \lfloor v \rfloor + \sum_{x \in C} \text{frac}(v(x)) \cdot e_{\{x\}}$. Suppose that we partition C in $r_0, \ldots, r_k$ such that if $x, y \in r_i$, then $\text{frac}(v(x)) = \text{frac}(v(y))$. Let $x_i \in r_i$, we can then establish the equality

$$v = \lfloor v \rfloor + \sum_{i=0}^{k} \text{frac}(v(x_i)) \cdot e_{r_i}$$

Definition 3.4 (Priced Regions) *Given set S, let Seq(S) be the set of finite sequences of elements of S. A* priced clock region *over a finite set of clocks C*

$$R = (h, [r_0, \ldots, r_k], [c_0, \ldots, c_k])$$

is an element of $(\mathbb{N}^C) \times$ Seq$(2^C) \times$ Seq$(\mathbb{N})$, with $C = \cup_{i \in \{0,\ldots,k\}} r_i$, $r_i \cap r_j = \emptyset$ when $i \neq j$, and $i > 0$ implies that $r_i \neq \emptyset$.

Given a clock valuation $v \in \mathbb{R}^C$, and region $R = (h, [r_0, \ldots, r_k], [c_0, \ldots, c_k])$, $v \in R$ iff

1. $h = \lfloor v \rfloor$,
2. $x \in r_0 \Leftrightarrow \text{frac}(v(x)) = 0$,
3. $x, y \in r_i \Leftrightarrow \text{frac}(v(x)) = \text{frac}(v(y))$, *and*
4. $x \in r_i$, $y \in r_j$ *and* $i < j \Leftrightarrow \text{frac}(v(x)) < \text{frac}(v(y))$.

For a priced region $R = (h, [r_0, \ldots, r_k], [c_0, \ldots, c_k])$, the first two components of the triple constitute an ordinary (unpriced) region $\hat{R} = (h, [r_0, \ldots, r_k])$. The closure of region $\hat{R}$ defines a k-dimensional convex polyhedron. The region itself is only closed if it is a singleton set, i.e. a vertex. In all other cases the region does not contain vertices or lower dimensional faces.

The vertex of the closure that is closest to the origin is defined by h. We denote this vertex with v_0. We can then reach the next vertex v_1 by moving one unit in the direction of set r_k, from there to vertex v_2 by moving one unit in the direction of r_{k-1}, etc. Hence, $v_{i+1} = v_i + e_{r_{k-i}}$, for $i = 0, \ldots, k-1$. The priced region

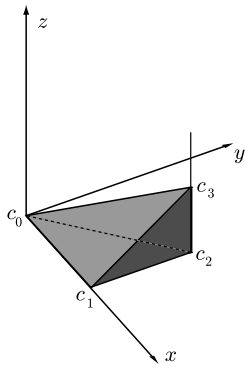


Figure 3.4: A three dimensional priced region.

R then assigns the costs $c_0, \ldots, c_k$ to the vertices $v_0, \ldots, v_k$, as depicted in Figure 3.4. The costs $c_0, \ldots, c_k$, span a linear cost plane on the k-dimensional unpriced region.

The closure of the unpriced region $\hat{R}$ is the convex hull of the vertices. Each clock valuation $v \in \hat{R}$ is a (unique) convex combination of the vertices. In particular $v = v_0 + \sum_{i=0}^{k-1} \text{frac}(v(x_{k-i}))(v_{i+1} - v_i)$, where x_j is some clock in r_j. This leads to the following definition:

Definition 3.5 (Cost inside Region) *Given priced region* $R = (h, [r_0, \ldots, r_k], [c_0, \ldots, c_k])$ *and clock valuation* $v \in R$*, the* cost *of* v *in* R *is defined as:*

$$\text{cost}(v, R) = c_0 + \sum_{i=0}^{k-1} \text{frac}(v(x_{k-i}))(c_{i+1} - c_i)$$

where x_j *is some clock in* r_j*. The minimal cost associated with* R *is* $\text{mincost}(R) = \min(\{c_0, \ldots, c_k\})$.

In the symbolic state-space, constructed with the priced regions, the costs will be computed such that for each concrete state in a symbolic state, the cost associated with it is the minimal cost for reaching that state by the symbolic path that was followed. In this way, we always have the minimal cost of all concrete paths represented by that symbolic path, but there may be more symbolic paths leading to a symbolic state in which the costs are different.

To prepare for the symbolic semantics, we define in the following a number of operations on priced regions. These operations are later used in the algorithm for finding the optimal cost of reaching a location.

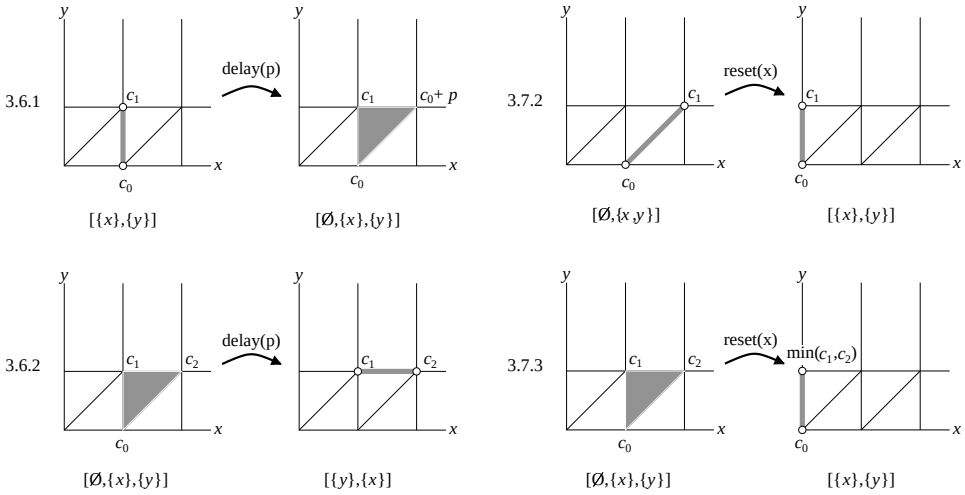


Figure 3.5: Delay and reset operations for two-dimensional priced regions.

Definition 3.6 (Delay) *Let $R = (h, [r_0, \ldots, r_k], [c_0, \ldots, c_k])$ be a priced region. Suppose that the set of clocks C is not empty. For a price p, the function* delay *is then defined as follows:*

1. *If $r_0 \neq \emptyset$, then*

$$\text{delay}(R, p) \quad = \quad (h, [\emptyset, r_0, \ldots, r_k], [c_0, \ldots, c_k, c_0 + p])$$

2. *If $r_0 = \emptyset$, then*

$$\begin{aligned} \text{delay}(R, p) \quad &= \quad (h', [r_k, r_1, \ldots, r_{k-1}], [c_1, \ldots, c_k]) \\ \textit{where } h' \quad &= \quad h + e_{r_k} \end{aligned}$$

The delay operation computes the time successor, and works exactly as for classical (unpriced) regions. The changing dimensions of the regions cause the addition or deletion of vertices and thus of the associated cost. The price argument will be instantiated to the price of the location in which time is passing; this is needed only if a vertex is added. The figures labeled (3.6.1) and (3.6.2) to the left of Figure 3.5 illustrate the two cases of definition 3.6.

Definition 3.7 (Reset) *Given a priced region $R = (h, [r_0, \ldots, r_k], [c_0, \ldots, c_k])$ and a clock $x \in r_i$, the function* reset *is defined as follows:*

1. *If* $i = 0$ *then*

$$\begin{aligned} \text{reset}(x, R) &= (h', [r_0, \ldots, r_k], [c_0, \ldots, c_k]), \\ \textit{where } h' &= h[x \mapsto 0] \end{aligned}$$

2. *If* $i > 0$ *and* $r_i \neq \{x\}$*, then*

$$\begin{aligned} \text{reset}(x, R) &= (h', [r_0 \cup \{x\}, \ldots, r_i \setminus \{x\}, \ldots, r_k], [c_0, \ldots, c_k]) \\ \textit{where } h' &= h[x \mapsto 0] \end{aligned}$$

3. *If* $i > 0$ *and* $r_i = \{x\}$*, then*

$$\begin{aligned} \text{reset}(x, R) &= (h', [r_0 \cup \{x\}, \ldots, r_{i-1}, r_{i+1}, \ldots, r_k], \\ &\qquad [c_0, \ldots, c_{k-i-1}, c', c_{k-i+2}, \ldots, c_k]) \\ \textit{where } c' &= \min(c_{k-i}, c_{k-i+1}) \\ h' &= h[x \mapsto 0] \end{aligned}$$

The reset operation on a set of clocks: $\text{reset}(C \cup \{x\}, R) = \text{reset}(C, \text{reset}(x, R))$*, and* $\text{reset}(\emptyset, R) = R$*.*

When resetting a clock, a priced region may loose a dimension. In this case, the two costs, associated with the vertices that are collapsed, are compared and the minimum is taken for the new vertex. The figures (4.7.2) and (4.7.3) to the right of Figure 3.5 illustrate the nontrivial cases 2 and 3 of the definition.

.

Definition 3.8 (Increment) *For a priced region* $R = (h, [r_0, \ldots, r_k], [c_0, \ldots, c_k])$ *and a price* p*, the increment of* R *with respect to* p *is the priced region* $R \oplus p = (h, [r_0, \ldots, r_k], [c'_0, \ldots, c'_k])$ *where* $c'_i = c_i + p$*.*

The price argument in the increment operation will be instantiated to the price of the particular transition taken; all costs are updated accordingly

If in region R, no clock has fractional part 0, then time may pass in R, that is, each clock valuation in R has a time successor and predecessor in R. When changing location with R, we must choose for each clock valuation v in R between delaying in the previous location until v is reached, followed by the change of location, or changing location immediately and delaying to v in the new location. Which is cheapest, depends on the price of either location. For this the following operation self is useful.

Definition 3.9 (Self) *Let $R = (h, [r_0, \ldots, r_k], [c_0, \ldots, c_k])$ be a priced region. Suppose that the set of clocks C is not empty. For a price p, the function* self *is defined as follows:*

1. *If r_0 is not empty, then* $\text{self}(R, p) = R$.
2. *If r_0 is empty, then*

$$\begin{aligned} \text{self}(R, p) &= (h, [r_0, \ldots, r_k], [c_0, \ldots, c_{k-1}, c']) \\ \textit{where } c' &= \min(c_k, c_0 + p) \end{aligned}$$

If the set C is empty, then $\text{self}(R, p) = R$.

Definition 3.10 (Comparison) *Let $R = (h, [r_0, \ldots, r_k], [c_0, \ldots, c_k])$ and $R' = (h', [r'_0, \ldots, r'_{k'}], [c'_0, \ldots, c'_{k'}])$ may be priced regions. We define*

$$R \leq R' \textit{ iff } h = h', k = k', \textit{ and for } 0 \leq i \leq k: \;\; r_i = r'_i \wedge c_i \leq c'_i$$

From the definition of a region and the cost inside a region follows, that if $R \leq R'$ and $v \in R$, then $v \in R'$ and $\text{cost}(v, R) \leq \text{cost}(v, R')$.

The operations delay, reset, $\oplus$ and self satisfy the following useful properties:

Proposition 3.11 (Interaction Properties)

1. $\text{self}(R, p) \leq R$,
2. $\text{self}(\text{self}(R, p), p) = \text{self}(R, p)$,
3. $R \leq R' \Rightarrow \text{delay}(R, p) \leq \text{delay}(R', p)$,
4. $\text{reset}(x, \text{reset}(x, R)) = \text{reset}(x, R)$,
5. $\text{reset}(x, \text{reset}(y, R)) = \text{reset}(y, \text{reset}(x, R))$,
6. $\text{self}(\text{delay}(R, p), p) = \text{delay}(R, p)$,
7. $\text{self}(R \oplus q, p) = \text{self}(R, p) \oplus q$,
8. $\text{delay}(R \oplus q, p) = \text{delay}(R, p) \oplus q$,
9. *For $g \in \mathcal{B}(C)$, whenever $R \in g$ then* $\text{self}(R, p) \in g$.

Proof Follows from the definitions of the operators and $\leq$. □

Proposition 3.12 (Cost Relations) Given a priced region $R = (h, [r_0, \ldots, l_k], [c_0, \ldots, c_k])$, and a price p, the symbolic operations behave as follows:

1. If $R = \text{self}(R, p)$, $v' \in \text{delay}(R, p)$ and $v' - d \in R$ then

$$\text{cost}(v', \text{delay}(R, p)) = \inf\{\text{cost}(v' - d, R) + d \cdot p | v' - d \in R,\ d \in \mathbb{R}_{\geq 0}\}$$

2. $\text{cost}(v', \text{reset}(x, R)) = \inf\{\, \text{cost}(v, R) \mid v[x \mapsto 0] = v' \,\}$.

3. $\text{cost}(v, R \oplus p) = \text{cost}(v, R) + p$

Proof

1. Let $R = (h, [r_0, \ldots, l_k], [c_0, \ldots, c_k])$ be a priced region.

   (a) First, assume $r_o \neq \emptyset$. The delay successor of R is then the priced region $R' = (h, [\emptyset, r_0, \ldots, r_k], [c_0, \ldots, c_k, c_0 + p])$. Recall that, if $v' - d \in R$, and $x \in r_0$, then $\text{frac}(v'(x) - d) = 0$. Note that for $v \in R$, $v + d \in R'$ and $x \in C$ holds $\text{frac}(v(x) + d) = \text{frac}(v(x)) + d$. We thus obtain

$$\begin{aligned}
\text{cost}(v', R') &= c_0 + \sum_{i=0}^{k-1} \text{frac}(v'(x_{k-i}) - d + d)(c_{i+1} - c_i) \\
&\quad + \text{frac}(v'(x_0) - d + d)(c_0 + p - c_k) \\
&= c_0 + \sum_{i=0}^{k-1} \text{frac}(v'(x_{k-i}) - d)(c_{i+1} - c_i) \\
&\quad + \text{frac}(v'(x_0) - d)(c_0 + p - c_k) && \{= 0\} \\
&\quad + d \sum_{i=0}^{k-1} (c_{i+1} - c_i) + d(c_0 + p - c_k) && \{= d \cdot p\} \\
&= \quad \text{cost}(v' - d, R) + d \cdot p
\end{aligned}$$

   (b) Next, assume $r_0 = \emptyset$. Since $R = \text{self}(R, p)$, we have $c_0 + p \geq c_k$. The delay successor of R is $R' = (h + e_{r_k}, [r_k, r_1, \ldots, r_{k-1}], [c_1, \ldots, c_k])$. If $v' \in R'$ and $0 < d < \text{frac}(v(x_1))$, then $v' - d \in R$. Note, that for $x_i \in r_i$, $v \in R$, and $v + d \in R'$ we have $\text{frac}(v(x_i) + d) =$

frac$(v(x_i))+d$ if $0<i<k$, and frac$(v(x_k))+d=1$. Hence, we have

$$\begin{aligned}\text{cost}(v',R') = &\quad c_0 + \sum_{i=1}^{k-1} \text{frac}(v'(x_{k-i}))(c_{i+1}-c_i) + 1\,(c_1-c_0)\\ = &\quad c_0 + \sum_{i=0}^{k-1} (\text{frac}(v'(x_{k-i})-d)+d)(c_{i+1}-c_i)\\ = &\quad \text{cost}(v'-d,R) + d(c_k-c_0)\\ \leq &\quad \text{cost}(v'-d,R) + d\cdot p\end{aligned}$$

For $d \to 0$, the last inequality becomes a equality, which proves Proposition 3.12.1. □

2. Let $R=(h,[r_0,\ldots,l_k],[c_0,\ldots,c_k])$ be a priced region, $x_j \in r_j$ and $R'=\text{reset}(R,x)$. For $i=0$ or $r_i \neq \{x_i\}$, it follows directly from the definitions of reset and cost that $\text{cost}(v,R)=\text{cost}(v[x_i\mapsto 0],R')$.

   Next, consider the case $0<i<k$ and $r_i=\{x_i\}$, then $R'=(h[x_i\mapsto 0],[r_0\cup\{x_i\},\ldots,r_{i-1},r_{i+1},\ldots,r_k],\ [c_0,\ldots,c_{k-i-1},c',c_{k-i+2},\ldots,c_k])$, with $c'=\min(c_{k-i},c_{k-i+1})$. Let $v'\in R'$. Clock valuations $v\in R$ satisfy $v[x\mapsto 0]=v'$, iff $v(x_j)=v'(x_j)$, for $i\neq j$ and $v'(x_{i-1})<v(x_i)<v'(x_{i+1})$. Since the cost is linear, the infimum will be attained in either of the vertices of this interval.

   The cost $\text{cost}(v,R)$ assigned to clock valuations $v\in R$ with $v[x_i\mapsto 0]=v'$, differ only in the summand $\text{frac}(v(x_i))(c_{k-i+1}-c_{k-i})$. If $c_{k-i+1}\geq c_{k-i}$, the infimum of $\text{cost}(v,R)$ will be attained for $v(x_i)\to v(x_{i-1})$. In this case the summands $\text{frac}(v(x_{i-1}))(c_{k-i+2}-c_{k-i+1})+\text{frac}(v(x_i))(c_{k-i+1}-c_{k-i})$ of $\text{cost}(v,R)$ simplify to $\text{frac}(v(x_{i-1}))(c_{k-i+2}-c_{k-i})$. Hence, we have $\text{cost}(v',R')=\inf\{\text{cost}(v,R)|v[x\mapsto 0]=v'\}$, as desired.

   Now, assume that $c_{k-i+1}\leq c_{k-i}$. The infimum of $\text{cost}(v,R)$, for $v[x_i\mapsto 0]=v'$, will then be attained for $v(x_i)\to v(x_{i+1})$. The two summands $\text{frac}(v(x_i))(c_{k-i+1}-c_{k-i})+\text{frac}(v(x_{i+1}))(c_{k-i}-c_{k-i-1})$ reduce in this case to $\text{frac}(v(x_{i+1}))(c_{k-i+1}-c_{k-i-1})$. This proves the equality $\text{cost}(v',R')=\inf\{\text{cost}(v,R)|v[x\mapsto 0]=v'\}$.

   Consider as last case $i=k$ and $r_k=\{x_k\}$. Clock valuations v satisfy $v[x_k\mapsto 0]=v'$, iff $v(x_j)=v'(x_j)$, for $j<k$ and $v'(x_{k-1})<v(x_k)<1$. Similarly to the previous case, the infimum will be attained in either vertex of this interval. The equality $\text{cost}(v',R')=\inf\{\text{cost}(v,R)|v[x\mapsto 0]=v'\}$ follows, similarly to the previous case, from the definition of cost in those vertices. □

3. Follows directly from the definitions of cost and $\oplus$. □

3.4 Symbolic Semantics

In this section, we provide a symbolic semantics for linearly priced timed automata based on the notion of priced regions and the associated operations presented in the previous section. As a main result we show that the cost of an execution of the underlying automaton is captured accurately. Finally, we present an algorithm based on priced regions.

Definition 3.13 (Symbolic Semantics) *The symbolic semantics of an LPTA A is defined as a transition system with the states Loc $\times$ ($\mathbb{N}^C \times$ Seq$(2^C) \times$ Seq$(\mathbb{N})$), with initial state $(l_0, (h_0, [C], [0]))$ (where h_0 assigns zero to all clocks in C), and with the following transition relation:*

- $(l, R) \rightarrow (l, \text{delay}(R, P(l)))$ *if* $\text{delay}(R, P(l)) \in$ *Inv*(l).
- $(l, R) \rightarrow (l', R')$ *if there exists g, a, r such that* $l \xrightarrow{g,a,r} l'$, $R \in g$, $R' = \text{reset}(r, R) \oplus P((l, g, a, r, l'))$, *and* $R' \in$ *Inv*(l').
- $(l, R) \rightarrow (l, \text{self}(R, P(l)))$.

In the remainder, states and executions of the symbolic transition system for LPTA A will be referred to as the symbolic states and executions of A.

Lemma 3.14 *Given LPTA A, for each execution α of A that ends in state (l, v), there is a symbolic execution β of A, that ends in symbolic state (l, R), such that $v \in R$, and* $\text{cost}(v, R) \leq \text{cost}(\alpha)$.

Proof First, observe that, given a constraint $g \in \mathcal{B}(C)$, if $v \in R$ and $v \in g$, then $R \in g$.

We prove lemma by induction on the length of α. The base step concerns α with length 0, consisting of only the initial state (l_0, v_0) where v_0 is the valuation assigning zero to all clocks. Clearly, $\text{cost}(\alpha) = 0$. Since the initial state of the symbolic semantics is the state $(l_0, (h_0, [C], [0]))$, in which h_0 assigns zero to the integer part of all clocks, and the fractional part of all clocks is zero, we can take β to be $(l_0, (h_0, [C], [0]))$. Clearly, there is only one valuation $v \in (h_0, [C], [0])$, namely the valuation v that assigns zero to all clocks, which is exactly v_0, and by definition, $\text{cost}(v_0, (h_0, [C], [0])) = 0$ and trivially $0 \leq 0$.

For the induction step, assume the following. We have an execution α in the concrete semantics, ending in (l, v), a corresponding execution β in the symbolic semantics, ending in (l, R), such that $v \in R$, and $\text{cost}(v, R) \leq \text{cost}(\alpha)$.

Suppose $\alpha' = \alpha \xrightarrow{a,p} (l', v')$. Then there is a transition $l \xrightarrow{a,g,r} l'$ in the automaton A such that $v \in g$, $v' = v[r \mapsto 0]$, $v' \in \mathit{Inv}(l')$ and $p = P((l, a, g, r, l'))$. Now $v \in g$ implies that $R \in g$. Let $R' = \text{reset}(r, R) \oplus p$. It is easy to show that $v' = v[r \mapsto 0] \in R'$ and as $v' \in R'$ we then have that $R' \in \mathit{Inv}(l')$. So $(l, R) \rightarrow (l', R')$. Hence we can extend β to $\beta' \rightarrow (l', R')$ such that

$$\begin{aligned} \text{cost}(v', R') &= \inf\{\, \text{cost}(v, R) \mid v[r \mapsto 0] = v' \,\} + p \\ &\leq \text{cost}(v, R) + p \\ &\leq^{\text{IH}} \text{cost}(\alpha) + p \\ &= \text{cost}(\alpha') \end{aligned}$$

as desired.

Suppose $\alpha' = \alpha \xrightarrow{\epsilon(d), p \cdot d} (l', v')$, where $p = P(l)$, i.e. $l' = l$, $v' = v + d$, and $v + e \in \mathit{Inv}(l)$ for $0 \leq e \leq d$. As for unpriced regions there exist sequences $R_0, R_1, \ldots, R_m$ and $d_1, \ldots, d_m$ of priced regions and delays such that $d = d_1 + \ldots + d_m$, $R_0 = R$ and for $i \in \{1, \ldots, m\}$, $R_i = \text{delay}(R_{i-1}, p)$ with $v + \sum_{k=1}^{i} d_k \in R_i$. This defines the sequence of regions visited without considering cost. To obtain the priced regions with the optimal cost we apply the self operation. Let $R'_0 = \text{self}(R_0, p)$ and for $i \in \{1, \ldots, m\}$ let $R'_i = \text{delay}(R'_{i-1}, p)$ (in fact, for $i \in \{1, \ldots, m\}$, $R'_i = \text{self}(R'_i, p)$ due to Proposition 3.11.6). Clearly we have the following symbolic extension of β:

$$\beta \rightarrow (l', R'_0) \rightarrow \ldots \rightarrow (l', R'_m)$$

By repeated application of Proposition 3.12.1 (the condition of Proposition 3.12.1 is satisfied for all $R'_i (i \geq 0)$ because of Proposition 3.11.6), and application of Proposition 3.11.1 we get:

$$\begin{aligned} \text{cost}(v', R'_m) &\leq \text{cost}(v, R'_0) + d \cdot p \\ &\leq \text{cost}(v, R) + d \cdot p \\ &\leq^{\text{IH}} \text{cost}(\alpha) + d \cdot p \\ &= \text{cost}(\alpha') \end{aligned}$$

□

Lemma 3.15 *Whenever (l, R) is a reachable symbolic state and $v \in R$, then* $\text{mincost}((l, v)) \leq \text{cost}(v, R)$.

Proof The proof is by induction on the length of the symbolic trace β reaching (l, R). In the base case, the length of β is 0 and $(l, R) = (l_0, R_0)$, where R_0 is the

initial price region $(h_0, [C], [0])$ in which h_0 associates 0 with all clocks. Clearly, there is only one valuation $v \in R_0$, namely the valuation which assigns 0 to all clocks. Obviously, mincost$((l_0, v_0)) = 0 \leq$ cost$(v_0, R_0) = 0$.

For the induction step, assume that (l, R) is reached by a trace β. Let $\beta' = \beta \rightarrow (l', R')$ be an extension of β. We consider three cases depending on the type of symbolic transition from (l, R) to (l', R'). Let $v' \in R'$.

Case 1: Suppose $(l, R) \rightarrow (l', R')$ is a symbolic delay transition. That is, $l' = l$, $R' =$ delay(R, p) with $p = P(l)$ and $R' \in$ *Inv*(l). We consider two sub-cases depending on whether the first set of clocks of R is empty or not.

Assume $R = (h, [r_0, \ldots, r_k], [c_0, \ldots, c_k])$ with $r_0 \neq \emptyset$. Then region $R' = (h, [\emptyset, r_0, \ldots, r_k], [c_0, \ldots, c_k, c_0 + p])$. Let $x \in r_0$ and let $v = v' - d$ where $d =$ frac$(v'(x))$. Then $v \in R$ and $(l, v) \xrightarrow{\epsilon(d), q} (l', v')$ where $q = d \cdot p$. Thus mincost$((l', v')) \leq$ mincost$((l, v)) + d \cdot p$. We have by induction hypothesis, mincost$((l, v)) \leq$ cost(v, R), and as cost$(v', R') =$ cost$(v, R) + d \cdot p$ (by case (a) of the proof of Proposition 3.12.1), we obtain, as desired, mincost$((l', v')) \leq$ cost(v', R').

Assume $R = (h, [r_0, r_1, \ldots, r_k], [c_0, \ldots, c_k])$ with $r_0 = \emptyset$. Thus $R' = (h + e_{r_k}, [r_k, r_1, \ldots, r_{k-1}], [c_1, \ldots, c_k])$. Now, let $x_i \in r_i$ with $0 < i \leq k$. Let $v' \in R'$. Clearly, for any d with $0 < d <$ frac$(v'(x_1))$ the following holds: $v' - d \in R$, and hence $(l, v' - d) \xrightarrow{\epsilon(d), p \cdot d} (l, v')$, which leads to

$$\begin{array}{rcl} \text{mincost}((l, v')) & \leq & \text{mincost}((l, v' - d)) + p \cdot d \\ & \leq^{\text{IH}} & \text{cost}(v' - d, R) + p \cdot d \end{array}$$

Note that frac$(v'(x_i) - d) =$ frac$(v'(x_i)) - d$ for $i = 1, \ldots, k-1$ and frac$(v'(x_k) - d) = 1 - d$. We obtain

$$\begin{array}{rcll} \text{cost}(v' - d, R) + p \cdot d = & c_0 + \sum_{i=1}^{k-1} (\text{frac}(v(x_{k-i})) - d)(c_{i+1} - c_i) & \\ & + (1 - d)(c_1 - c_0) + p \cdot d & \\ \longrightarrow & c_1 + \sum_{i=1}^{k-1} \text{frac}(v(x_{k-i}) - d)(c_{i+1} - c_i) & \{d \rightarrow 0\} \\ = & \text{cost}(v', R') & \end{array}$$

Thus mincost$((l', v')) \leq$ cost(v', R') as desired.

Case 2: Suppose $(l, R) \rightarrow (l', R')$ is a symbolic action transition. That is $R' =$ reset$(r, R) \oplus p$ for some transition $l \xrightarrow{g, a, r} l'$ of the automaton with $R \in g$ and

$p = P((l,g,a,r,l'))$. Now let $v \in R$ such that $v[r \mapsto 0] = v'$. Then clearly $(l,v) \xrightarrow{a,p} (l',v')$. Thus:

$$\begin{aligned}\text{mincost}((l',v')) &\leq \inf\{\,\text{mincost}((l,v)) + p \mid v \in R, v[r \mapsto 0] = v'\,\}\\ &\leq^{\text{IH}} \inf\{\,\text{cost}(v,R) \mid v[r \mapsto 0] = v'\,\} + p\\ &= \text{cost}(v', \text{reset}(r,R)) + p\\ &= \text{cost}(v', R') \quad \text{by Proposition 3.12.2 and 3.12.3}\end{aligned}$$

Case 3: Suppose $(l,R) \to (l',R')$ is a self transition. Thus, in particular $l' = l$. If $R = (h,[r_0,\ldots,r_k],[c_0,\ldots,c_k])$ with $r_0 \neq \emptyset$, then $R' = R$. The lemma follows immediately by applying the induction hypothesis to (l',R').

Otherwise, if $r_0 = \emptyset$, then R' and R are identical except for the cost of the 'last' vertex; i.e. $R = (h,[r_0,\ldots,r_k],[c_0,\ldots,c_{k-1},c_k])$ and $R' = (h,[r_0,\ldots,r_k],[c_0,\ldots,c_{k-1},c_0+p])$ with $c_0 + p < c_k$ and $p = P(l)$. Now let $x_i \in r_i$. Let $v' \in R'$. Clearly, for any d with $0 \leq d < \text{frac}(v'(x_1))$ we have $v' - d \in R'$ (and $v' - d \in R$) and $(l, v'-d) \xrightarrow{\epsilon(d),p\cdot d} (l,v')$. Now:

$$\begin{aligned}\text{mincost}((l,v')) &\leq \text{mincost}((l,v'-d)) + p\cdot d\\ &\leq^{\text{IH}} \text{cost}(v'-d,R) + p\cdot d\end{aligned}$$

Observe that for $0 \leq d < \text{frac}(v(x_1))$ we have $\text{frac}(v(x) - d) = \text{frac}(v(x)) - d$ for all $x \in C$. Then

$$\begin{aligned}\text{cost}(v'-d,R) + p\cdot d = \quad & c_0 + \sum_{i=0}^{k-2} \text{frac}(v'(x_{k-i}))(c_{i+1} - c_i)\\ & + \text{frac}(v'(x_1))(c_k - c_{k-1}) + d(c_0 - c_k + p)\\ \longrightarrow\ & c_0 + \sum_{i=0}^{k-2} \text{frac}(v'(x_{k-i}))(c_{i+1} - c_i)\\ & + \text{frac}(v'(x_1))(c_0 + p - c_{k-1}) \ \ \{d \to \text{frac}(v'(x_1))\}\\ = \quad & \text{cost}(v',R')\end{aligned}$$

Hence, as desired, $\text{mincost}((l',v')) \leq \text{cost}(v',R')$. □

Combining the two lemmas we obtain as a main theorem that the symbolic semantics captures (sufficiently) accurately the cost of reaching states and locations. Lemma 3.15 ensures that we can find for any symbolic execution to (l,R) a reachable state (l,v), with $v \in R$, in the concrete semantics with less or equal cost. For this state however Lemma 3.14 ensures that we can find a symbolic trace to (l,R') with less cost and $v \in R'$. This allows us to establish the following equality.

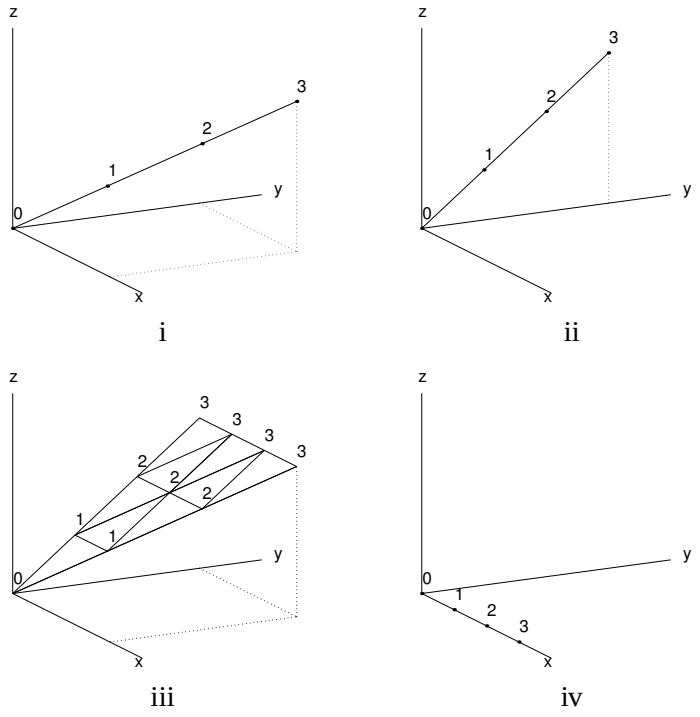


Figure 3.6: Sets of reachable priced regions of the LPTA in Figure 3.2.

Theorem 3.16 *Let l be a location of a LPTA A. Then*

$$\text{mincost}(l) = \min(\{\, \text{mincost}(R) \mid (l, R) \textit{ is reachable} \,\})$$

3.5 Example Symbolic State-Space

In this section, we present part of the symbolic state-space of the linearly priced timed automaton in Figure 3.2 where the value of both α and β is two. Figures 3.6.i-iv and 3.7.v-viii show some of the priced regions reachable in a symbolic representation of the states space. Regions are the vertices, edges and the interior of the triangles. We only show the priced regions with integer value less than or equal to three.

Initially all three clocks have value zero and when delaying the clocks keep on all having the same value. Therefore the priced regions reachable from the initial state by the delay operation are the ones on the line from $(0, 0, 0)$ through $(3, 3, 3)$ shown in Figure 3.6.i. The numbers on the line are the costs of the vertices of

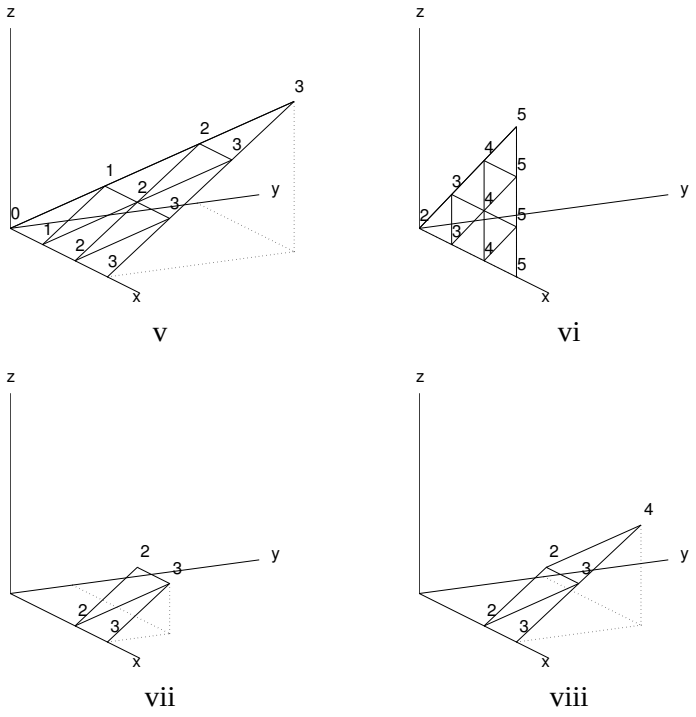


Figure 3.7: Sets of reachable priced regions (continued).

the priced regions. Since the cost of staying in location A is one, the price of delaying one time unit is one. The point $(3, 3, 3)$ is the only element of region $((3, 3, 3), [\{x, y, z\}], 3)$.

The transition labeled a from location A to B resets clock x. All regions in Figure 3.6.i can take this transition, since the guard is trivially true. This yields the priced regions presented in Figure 3.6.ii. The reset does not change any of the costs, since the new priced regions are still one-dimensional and no vertices are collapsed. The priced regions in Figure 3.6.iii are the delay successors of the priced regions in Figure 3.6.ii.

The transition c from location B to location C resets clocks y and z. After resetting the priced regions in Figure 3.6.iii, the priced regions in Figure 3.6.iv are reachable. The cost of a state s after the reset (projection) is the infimum of the cost of the states projecting to s. But since the cost are linear it is sufficient to consider in this case only the cost in the vertices along the diagonal.

When delaying from these priced regions, the priced regions in Figure 3.7.v

are reached. Now we are left with a choice; either we can take the transition d to location D, or take the loop transition c. Transition c resets the y clock and has a price of $\alpha = 2$. Resetting clock y and adding 2 to the cost of the priced regions in Figure 3.7.v gives the regions in Figure 3.7.vi. Taking transition c again yields the same priced regions as displayed in Figure 3.7.vi but now with just another two added to the cost. So, for each successor R' of a region R depicted in Figure 3.7.vi we have $R \leq R'$. Therefore the new priced regions are more costly than the priced regions already found. The algorithm presented in the next section will decide with this information not to explore this state further.

Taking the transition to location D is only possible if the guard $x \geq 2 \wedge y \leq 1$ is satisfied. Only the priced regions in Figure 3.7.vii are reachable from the regions in 3.7.v by transition d. The minimum cost of reaching location D in this way is two. The cost rate in location D is $\beta = 2$, but the self transition does not change any of these regions. Application of delay operations gives then the regions in Figure 3.7.viii. The delay successor of region $((2,1,1),[\{y,z\},\{x\}],[2,3])$ for instance is $((2,1,1),[\emptyset,\{y,z\},\{x\}],[2,3,4])$. While the cost increase in the old regions increases with rate 1, the cost of regions that are only reachable by a delay in location D increases with rate 2.

3.6 Algorithm

The introduction of priced regions provides a first step towards an algorithmic solution for the minimum-cost reachability problem. In the present form, however, both the integral part as well as the cost of vertices of priced regions may grow beyond any given bound during symbolic exploration. In the unpriced case, the growth of integral parts is often dealt with by suitable abstractions of (unpriced) regions, taking the maximal constant of the given timed automaton into account.

We have chosen a very similar approach exploiting the fact, that we do not care about the exact value of clocks that exceed some bound beyond the maximal constant. As soon as it reaches this bound, we reset it to a smaller bound which is also larger than the maximal constant. Of course, this requires a more general definition of linearly priced timed automata, that allows to reset a clock not just to 0, but to an integer value beyond the maximal constant $\max_A$. But this extension is obtained straightforward and requires only slight modifications of the LPTA model, in particular of Definition 3.7.

This section shows that any LPTA A may be transformed into an equivalent *bounded* automaton $\tilde{A}$ in the sense that A and $\tilde{A}$ reach the same locations with the exact same cost. We then present a state-space exploration algorithm and show that it computes the optimal trace, provided that it terminates. We then show that the

algorithm terminates for any bounded LPTA, and thus for any LPTA.

Theorem 3.17 *Let $A = (Loc, l_0, E, Inv, P)$ be a LPTA with maximal constant* max. *Then there exists a* bounded time equivalent *of A, $\tilde{A} = (Loc, l_0, E \cup E', Inv', P')$, satisfying the following:*

1. *Whenever (l, v) is reachable in $\tilde{A}$, then for all $x \in C$, $v(x) \leq \max_A + 2$.*
2. *For any location $l \in Loc$, l is reachable with cost c in A if and only if l is reachable with cost c in $\tilde{A}$*

Proof We construct $\tilde{A} = (Loc, l_0, E \cup E', Inv', P')$, as follows. $E' = \{(l, x == \max + 2, \tau, x := \max_A + 1, l) | x \in C, l \in Loc\}$. For $l \in Loc$, $Inv'(l) = Inv(l) \wedge \bigwedge_{x \in C} x \leq \max_A + 2$, $P'(l) = P(l)$. For $e \in (E \cup E')$, if $e \in E$ then $P'(e) = P(e)$ else $P'(e) = 0$.

By definition, $\tilde{A}$ satisfies the first requirement.

As to the second requirement. Let R be a relation between states from A and $\tilde{A}$ such that for $((l_1, v_1), (l_2, v_2)) \in R$ iff $l_2 = l_1$, and for each $x \in C$, if $v_1(x) \leq \max_A$ then $v_2(x) = v_1(x)$, else $v_2(x) > \max_A$. We show that for each state (l_1, v_1) of A which is reached with cost c, there is a state (l_2, v_2) of $\tilde{A}$, such that $((l_1, v_1), (l_2, v_2)) \in R$ and (l_2, v_2) is reached with cost c, and vice versa.

Let (l_1, v_1), (l_2, v_2) be states of A and $\tilde{A}$, respectively. We use induction on the length of some execution leading to (l_1, v_1) or (l_2, v_2). For the base step, the length of such an execution is 0. Trivially, the cost of such an execution is 0, and if (l_1, v_1) and (l_2, v_2) are initial states, clearly $((l_1, v_1), (l_2, v_2)) \in R$.

For the transition steps, we first observe that for each clock $x \in C$,

$$v_1(x) \sim c \quad \text{iff} \quad v_2(x) \sim c \text{ with } \sim \in \{<, \leq, >, \geq\} \text{ and } c \leq \max_A. \quad (*)$$

Assume $((l_1, v_1), (l_2, v_2)) \in R$, and (l_1, v_1) and (l_2, v_2) can both be reached with cost c. We make the following case distinction.

Case 1: Suppose $(l_1, v_1) \xrightarrow{\epsilon(d), p}_A (l_1, v_1 + d)$. In order to let d time pass in (l_2, v_2), we have to alternatingly perform the added τ transition to reset those clocks that have reached the $\max_A + 2$ bound as many times as needed, and then let a bit of the time pass. Let $d_1 \ldots d_m$ be a sequence of delays, such that $d = d_1 + \ldots + d_m$, and for $x \in C$ and $i \in \{1, \ldots, m\}$, if $\max_A + 2 - (v_1(x) + d_1 + \ldots + d_{i-1}) \geq 0$ then $d_i \leq \max_A + 2 - (v_1(x) + d_1 + \ldots + d_{i-1})$, else $d_i \leq 1 - \text{frac}(v_1(x) + d_1 + \ldots + d_{i-1})$. It is easy to see that for some v_2',

$$(l_2, v_2)(\xrightarrow{\tau, 0})^* \xrightarrow{\epsilon(d_1), p_1} \ldots (\xrightarrow{\tau, 0})^* \xrightarrow{\epsilon(d_m), p_m} (l_2, v_2')$$

where $p_i = d_i \cdot P(l_2)$. The cost for reaching $(l_1, v_1 + d)$ is $c + d \cdot P_A(l_1) = c + d \cdot P_{\tilde{A}}(l_2) = c + (d_1 + \ldots + d_m) \cdot P_{\tilde{A}}(l_2)$, which is the cost for reaching (l_2, v_2').

Now, $((l_1, v_1 + d), (l_2, v_2')) \in R$, because of the following. For each $x \in C$, if $v_1(x) > \max_A$, then $v_2(x) > \max_A$, and either x is not reset to $\max_A + 1$ by any of the τ transitions, in which case still $v_2'(x) > \max_A$, or x is reset by some of the τ transitions, and then $\max_A + 1 \leq v_2'(x) \leq \max_A + 2$, so $v_2'(x) > \max_A$.

If $v_1(x) \leq \max_A$, then by $v_1(x) = v_2(x)$, $v_2(x) \leq \max_A$. If $(v_1 + d)(x) \leq \max_A$, then x is not touched by any of the τ transitions leading to (l_2, v_2'), hence $v_2'(x) = v_2(x) + d_1 + \ldots + d_m = v_2(x) + d = (v_1 + d)(x)$. If $(v_1 + d)(x) > \max_A$, then x may be reset by some of the τ transitions leading to (l_2, v_2'). If so, then $\max_A + 1 \leq v_2'(x) \leq \max_A + 2$, so $v_2'(x) > \max_A$. If not, then $v_2'(x) = v_2(x) + d_1 + \ldots + d_m = v_2(x) + d = (v_1 + d)(x) > \max_A$.

Case 2: Suppose $(l_2, v_2) \xrightarrow{\epsilon(d),p}_A (l_2, v_2 + d)$. Then it trivially holds that $((l_1, v_1 + d), (l_2, v_2 + d)) \in R$. Now we show $(l_1, v_1) \xrightarrow{\epsilon(d),p}_A (l_1, v_1 + d)$. Since $(l_2, v_2 + d) \in \mathit{Inv}_{\tilde{A}}$, since $\mathit{Inv}_{\tilde{A}}$ implies Inv_A and since $((l_1, v_1 + d), (l_2, v_2 + d)) \in R$, from observation $(*)$ it follows that $(l_1, v_1 + d) \in \mathit{Inv}_A$. So $(l_1, v_1) \xrightarrow{\epsilon(d),p}_A (l_1, v_1 + d)$, and trivially, the cost of reaching $(l_2, v_2 + d)$ is $c + d \cdot P_{\tilde{A}}(l_2) = c + d \cdot P_A(l_1)$, which is the cost of reaching $(l_1, v_1 + d)$.

Case 3: Suppose $(l_1, v_1) \xrightarrow{a,p}_A (l_1', v_1')$. Let (l, g, a, r, l') be a corresponding edge. Then $p = P_A((l, g, a, r, l'))$. By definition, $(l, g, a, r, l') \in E_{\tilde{A}}$ and $P_{\tilde{A}}((l, g, a, r, l')) = P_A((l, g, a, r, l'))$. From observation $(*)$ it follows that $v_1 \in g$ implies $v_2 \in g$. It is easy to see that for $x \in r$, $v_1'(x) = 0 = v_2[r \mapsto 0](x)$, and for $x \notin r$, $v_1'(x) = v_1(x)$ and $v_2(x) = v_2[r \mapsto 0](x)$, so $((l_1', v_1'), (l', v_2[r \mapsto 0])) \in R$. Combining this with observation $(*)$ it follows that $v_1[r \mapsto 0] \in \mathit{Inv}_A(l')$ implies $v_2[r \mapsto 0] \in I_{\tilde{A}}(l')$, hence $(l_2, v_2) \xrightarrow{a,p}_{\tilde{A}} (l', v_2[r \mapsto 0])$. Clearly, the cost of reaching (l_1, v_1') is $c + d \cdot P_{\tilde{A}}((l, g, a, r, l')) = c + d \cdot P_A((l, g, a, r, l'))$, which is the cost of reaching $(l_2, v_2[r \mapsto 0])$.

Case 4: Suppose $(l_2, v_2) \xrightarrow{a,p}_{\tilde{A}} (l_2', v_2')$. Let (l, g, a, r, l') be a corresponding edge. If $(l, g, a, r, l') \in E_A$, then the argument goes exactly like in the previous case. If $(l, g, a, r, l') \notin E_A$, then $a = \tau$, $p = 0$, $l_2' = l' = l = l_2$, and $x \in r$ implies $v_2'(x) = \max_A + 1$ and $v_2(x) = \max_A + 2$. Since the cost of reaching (l_2', v_2') is $c + 0 = c$, it suffices to show $((l_1, v_1), (l_2, v_2')) \in R$. For $x \notin r$, this follows trivially. For $x \in r$, $v_2(x) = \max_A + 2$, so $v_1(x) > \max_A$ and by $v_2'(x) = \max_A + 1$ we have $v_2'(x) > \max_A$. □

Now, we suggest in Figure 3.8 a branch-and-bound algorithm for determining the minimum-cost of reaching a given target location l_g from the initial state of a LPTA. All encountered states are stored in the two data structures PASSED and WAITING, divided into explored and unexplored states, respectively. The global variable COST stores the lowest cost for reaching the target location found so far.

```
COST := ∞
PASSED := {}
WAITING := [(l_0, R_0)]
while WAITING ≠ [] do
   select (l, R) from WAITING
   if l = l_g and mincost(R) < COST
   then COST := mincost(R)
   if ∀(l', R') ∈ PASSED. l ≠ l' ∨ R' ⊈ R
   then add (l, R) to PASSED
        forall (l', R') s.t. (l, R)→(l', R') do
           add (l', R') to WAITING
        od
   fi
od
```

Figure 3.8: Branch-and-bound state-space exploration algorithm.

In each iteration, a state is taken from WAITING. If it matches the target location l_g and has a lower cost than the previously lowest cost COST, then COST is updated. Then, only if the state has not been previously explored with a lower cost we add it to PASSED and its successors to WAITING. This bounding of the search in line 8 of Figure 3.8 may be optimized even further by adding the constraint mincost$(R) <$ COST; i.e. we only need to continue exploration if the minimum cost of the current region is below the optimal cost computed so far. Due to Theorem 3.16, the algorithm of Figure 3.8 does indeed yield the correct minimum-cost value, provided that it terminates.

Theorem 3.18 *When the algorithm in Figure 3.8 terminates, the value of* COST *equals* mincost(l_g).

Proof First, notice that if (l_1, R_1) can reach (l_2, R_2) via a sequence of transitions, then a state (l_1, R_1') with $R_1' \leq R_1$ can reach a state (l_2, R_2') with $R_2' \leq R_2$ via the same path. We prove that after termination COST equals $\min\{\text{mincost}(R) \mid (l_g, R) \text{ is reachable}\}$.

The case COST $\geq \min\{\text{mincost}(R) \mid (l_g, R) \text{ is reachable}\}$ is true, since the value of COST is only updated if (l_g, R) is found to be reachable. The case COST $\leq \min\{\text{mincost}(R) \mid (l_g, R) \text{ is reachable}\}$ is less trivial. If the minimum exists, there has to be an execution $(l_0, R_0) \rightarrow \ldots \rightarrow (l_n, R_n)$, with $l_n = l_g$ and mincost$(R) = \min\{\text{mincost}(R) \mid (l_g, R) \text{ is reachable}\}$. If this state is explored COST will be updated to mincost(R_n), and we are done.

If this is not the case the algorithm must at some point have discarded a symbolic state (l_i, R_i) on the path to (l_n, R_n). This can only happen in line 8, but then

there must exist a state $(l'_i, R'_i) \in$ PASSED with $R'_i \leq R_i$, which was encountered in a prior iteration of the loop. Hence, there is a state (l'_n, R'_n) reachable via execution fragment $(l'_i, R'_i) \rightarrow \ldots \rightarrow (l'_n, R'_n)$, such $l'_n = l_g$ and mincost$(R'_n) \leq$ mincost(R_n). The last inequality is actually an equality, since mincost(R_n) is chosen minimal. Note that the execution fragment $(l'_i, R'_i) \rightarrow \ldots \rightarrow (l'_n, R'_n)$ has the same length as $(l_i, R_i) \rightarrow \ldots \rightarrow (l_n, R_n)$. If the algorithm explores symbolic state (l'_n, R'_n) we have COST $\leq$ mincost(R'_n) as desired.

If this is not the case one of the states (l'_j, R'_j) with $i < j < n$ has been discarded. In this case there must exist $(l''_j, R''_j) \in$ PASSED with $R''_j \leq R'_j$, but also a path $(l''_j, R''_j) \rightarrow \ldots \rightarrow (l''_n, R''_n)$ such that $l''_n = l_g$ and mincost$(R''_n) =$ mincost(R_n). We are now in the same situation as in the previous paragraph, except for the length of execution fragment to the goal location, which decreased. If we iterate this step, the distance to the goal location will decrease with each iteration, and we will eventually (after at most $n - i$ steps) find a symbolic state (l_g, R) that has been explored by the algorithm. Thus COST $\leq$ mincost$(R) =$ $\min\{$mincost$(R) \mid (l_g, R)$ is reachable$\}$.

The theorem now follows from Theorem 3.16. □

For bounded LPTA, application of Dickson's Lemma, which is a special case of Higman's Lemma [Hig52], ensures termination. In [Vel00] Dickson's Lemma is formulated as follows:

Lemma 3.19 (Dickson) *For every $p > 0$, for all infinite sequences $\alpha_0, \ldots, \alpha_{p-1}$ of natural numbers, there exist i, j such that $i < j$ and for every $k < p$: $\alpha_k(i) \leq \alpha_k(j)$.*

We will use this lemma to prove the following:

Theorem 3.20 *The algorithm in Figure 3.8 terminates for any bounded LPTA.*

Proof Even if A is bounded (and hence yields only finitely many unpriced regions), there are still infinitely many priced regions, due to the unboundedness of cost of vertices. Application of Dickson's lemma ensures, since all costs are positive, that one cannot have an infinite sequence $\langle (c^i_1, \ldots, c^i_m) \ : \ 0 \leq i < \infty \rangle$ of cost-vectors (for any fixed length m) without $c^j_l \leq c^k_l$ for all $l = 1, \ldots, m$ for some $j < k$. Consequently, due to the finiteness of the sets of locations and unpriced regions, it follows that one cannot have an infinite sequence $\langle (l_i, R_i) \ : \ 0 \leq i < \infty \rangle$ of symbolic states without $l_j = l_k$ and $R_j \leq R_k$ for some $j < k$, thus ensuring termination of the algorithm. □

Finally, combining Theorem 3.18 and 3.20, it follows, due to Theorem 3.17, that the minimum-cost reachability problem is decidable.

Theorem 3.21 *The minimum-cost problem for LPTA is decidable.*

3.7 Conclusion

In this chapter, we have successfully extended the work on regions and their operations to a setting of timed automata with linear prices on both transitions and locations. We have given an elementary principle branch-and-bound algorithm for the minimum-cost reachability problem, which is based on accurate symbolic semantics of timed automata with linear prices, and thus showing the minimum-cost reachability problem to be decidable.

A slight modification of our algorithm provides an extension to a parameterized setting, in which (some) prices may be parameters. In this setting, costs within priced regions are linear expressions over the given parameters rather than simple natural numbers. From recent results in [AN00] (generalizing Higman's lemma) it follows that the ordering on (parameterized) symbolic states is again a well-quasi ordering, hence guaranteeing termination of our algorithm. In the modified version of algorithm, COST will be a collection of (linear) cost-expressions with which the goal-location has been reached (so far).

The algorithm in Figure 3.8 is guaranteed to be rather inefficient and highly sensitive to the size of constants used in the guards of the automata — a characteristic inherited from the unpriced regions. An obvious continuation of this work is therefore to investigate if other more efficient (in practice) data structures can be found. Possible candidates include data structures used in reachability algorithms of timed automata, such as DBMs to represent zones (i.e. convex sets of clock assignments). In contrast to the priced extension of regions, operations on such a notion of priced zones cannot be obtained as direct extensions of the corresponding operations on zones with suitable manipulation of cost of vertices. Consider for example Figure 3.7.viii. The regions form a zone, but the cost does not constitute a linear plane on this zone.

The next chapter will present an efficient algorithm for the sub class of *Uniformly Priced Timed Automata*. For this class we can basically use an implementation based on DBMs. Chapter 5 addresses the problem of symbolic exploration based on priced zones for the full class of *Linearly Priced Automata*.

4

Efficient Guiding for Uniformly Priced Timed Automata

4.1 Introduction

The previous chapter presented an algorithm for computing the minimal cost of reaching designated goal states for the full model of Linearly Priced Timed Automata (LPTA). Since the algorithm is based on a cost-extended version of regions it is guaranteed to be extremely inefficient and highly sensitive to the size of constants used in the models. This chapter presents an algorithm for computing the minimum cost of reaching a goal state in the model of *Uniformly Priced Timed Automata* (UPTA) a sub-class of the LPTA model. This algorithm is based on a symbolic semantics of UPTAs, and an efficient representation and operations based on difference bound matrices.

As pointed out in the conclusions of Chapter 2, using a verification algorithm to solve scheduling problems has several drawbacks. Verification algorithms do normally not support any notion of optimality and are designed to explore the entire state-space as efficiently as possible. The verification algorithms that do support notions of optimality are restricted to simple trace properties such as shortest trace [LPY95], or shortest accumulated delay in trace [NTY00]. Dedicated scheduling algorithms, in contrast, are often designed to find optimal (or near optimal) solutions and are therefore based on techniques such as branch-and-bound to identify and prune parts of the states-space that are guaranteed to not contain any optimal solutions.

This chapter is based on the publication:

[BFH+01a] G. Behrmann, A. Fehnker, T.S. Hune, K.G. Larsen, P. Petterson and J.M.T. Romijn. *Efficient Guiding Towards Cost-Optimality in* UPPAAL. TACAS'01.

It contains excerpts from the paper:

[BMF02] E. Brinksma, A. Mader and A. Fehnker. *Verification and Optimization of a PLC Control Schedule* STTT. 2002.

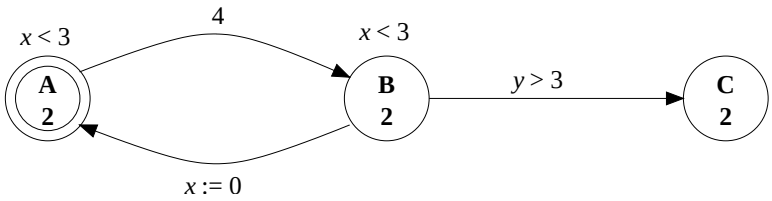


Figure 4.1: Example UPTA with two clocks x and y, and a uniform cost rate 2

In this chapter we aim at reducing the gap between scheduling and verification algorithms by adapting a number of techniques used in scheduling algorithms to timed automata and adding them to the verification tool UPPAAL. The modified model checker can then be used to solve the minimal-cost reachability problem for *Uniformly Priced Timed Automata*. UPTAs are LPTAs with the same cost rate in all locations. Delay does therefore contribute in a uniform manner to the cost, independent of the location. An example of a UPTA is depicted in Figure 4.1. The automaton is initially in location A. The transition from A to B has cost 4, whereas the other two transitions are free. Because of the invariants on the locations, a trace reaching the location C must first visit B and then go back to the initial location A. It can then reach location C, via B, with minimal cost of 14. UPTAs cover scheduling problems in which the overall time has to be minimized, like the Sidmar steel plant scheduling problem and job shop scheduling problems (Chapter 2).

As the first contribution of this chapter, we give for the sub-class of UPTA an efficient zone representation of symbolic cost states based on *Difference Bound Matrices* [Dil89], and give all the necessary symbolic operators needed to implement a state-space exploration algorithm that computes the optimal solution. As the second contribution we show, in analogy with Dijkstra's shortest path algorithm, that the state-space exploration may terminate as soon as a goal state is explored, if the algorithm is modified to always select the symbolic state with the smallest minimum cost from the WAITING list. This means that we can solve the minimum-cost reachability problem without necessarily searching the entire state-space of the analyzed automaton.

The third contribution of this chapter is a number of techniques inspired by branch-and-bound algorithms (page 40) that have been adopted in making the algorithm even more useful. These techniques are particularly useful for limiting the search space and for quickly finding solutions near to the minimum cost of reaching a goal state. To support this claim, we have implemented the algorithm in an experimental version of the verification tool UPPAAL and applied it to a wide variety of examples. Our experimental findings indicate that in some cases as much as

90% of the state-space searched in ordinary breadth-first order can be avoided by combining the techniques presented in this chapter. Moreover, the techniques allow reachability analysis to be performed in cases which were previously unsuccessful.

The rest of this chapter is organized as follows: Section 4.2 introduces the notion of a priced zone. In Section 4.3 we formally define the model of uniformly priced timed automata and give the symbolic semantics and present the basic algorithm. Section 4.4 discusses some useful properties of minimal-cost orders and modifications of the basic algorithm, inspired by branch-and-bound techniques. The experiments are presented in Section 4.5. We conclude the chapter in Section 4.6.

4.2 Priced Zones

The semantics of LPTA as defined in Section 3.2 yields an uncountable state-space and is therefore not suited for state-space exploration algorithms. For priced regions we have obtained a decidability result, but this approach suffers from the inefficiency it inherited from the unpriced regions. For pure reachability analysis, timed automata model checkers like UPPAAL and KRONOS use symbolic semantics based on *zones*. A zone $Z \subseteq \mathbb{R}^C_{\geq 0}$ is a convex set of clock valuations that satisfy a constraint from $\mathcal{B}(C)$, and can be considered as convex union of regions (Section 3.3. In the remainder we will identify the set of clock valuation with the constraints that define the set. Recall that $\mathcal{B}(C)$ is the set of conjunctions of atomic clock constraints of the form $x \sim n$ for $x, y \in C$, $n \in \mathbb{N}$ and $\sim \in \{<, \leq, =, \geq, >\}$, given a set of clocks C. Since each timed automaton with constraints of the form $x - y \sim n$, is strongly bisimilar to a timed automaton with only constraints of the form $x \sim n$, the results in this chapter are applicable to automata having constraints of the type $x - y \sim n$ as well.

Whenever Z is a zone and r a set of clocks, we denote by $Z^{\uparrow}$ and $\{r\}Z$ the set of clock valuations obtained by delaying and resetting (w.r.t. r) clock valuations from Z, respectively. That is, $Z^{\uparrow} = \{v + d \,|\, v \in Z, d \in \mathbb{R}_{\geq 0}\}$ and $\{r\}Z = \{v[r \mapsto 0] \,|\, v \in Z\}$. It is well-known – using a canonical representation of zones as *Difference Bounded Matrices* (DBMs) [Dil89] – that in both cases the resulting set is again representable as a zone. Using these operations together with the obvious fact, that zones are closed under conjunction, necessary operations may now be effectively realized using the zone-based representation of symbolic states.

In the priced setting we must in addition represent the costs with which individual states are reached. For this we propose *priced zones*:

Definition 4.1 (Priced Zone) *A priced zone $\mathcal{Z}$ is a tuple (Z, π), where Z is a zone and $\pi : Z \to \mathbb{R}_{\geq 0}$ a function from valuations in Z to real valued cost.*

The intention is that reachability of the priced symbolic state $(l, \mathcal{Z})$ should ensure that any state (l, v) with $v \in \mathcal{Z}$ is reachable with cost arbitrarily close to $\pi(s)$. For convenience we define for $v \in \mathbb{R}^C_{\geq 0}$ and $\mathcal{Z} = (Z, \pi)$ that cost$(v, \mathcal{Z})$ equals $\pi(v)$, if $v \in Z$ and ∞ otherwise. Given a clock valuation $v \in \mathbb{R}^C_{\geq 0}$, and a priced zone $\mathcal{Z} = (Z, \pi)$, we define $v \in \mathcal{Z}$ iff $v \in Z$.

As in the case of priced regions, we have to define comparison of priced zones. Let (Z, π) and (Z', π') be priced zones. We write $(Z, \pi) \sqsubseteq (Z', \pi')$ if $Z' \subseteq Z$ and $\pi(v) \leq \pi'(v)$ for all $v \in Z'$, informally expressing, that (Z, π) is "as big and cheap" as (Z', π'). Alternatively we may define $\mathcal{Z} \sqsubseteq \mathcal{Z}'$ iff cost$(v, \mathcal{Z}) \leq$ cost$(v, \mathcal{Z}')$, for all $v \in \mathbb{R}^C_{\geq 0}$. We denote the infimum $\inf\{\text{cost}(v, \mathcal{Z}) | v \in \mathbb{R}^C_{\geq 0}\}$ by mincost$(\mathcal{Z})$.

The symbolic semantics for LPTA based on priced zones is very similar to the common zone based symbolic semantics used for timed automata.

Definition 4.2 (Zone Semantics) *Let $A = (Loc, l_0, E, Inv, P)$ be a linearly priced timed automaton. The symbolic semantics is defined as a labeled transition system over symbolic states of the form $(l, \mathcal{Z})$, l being a location and $\mathcal{Z} = (Z, \pi)$ a priced zone with the transition relation:*

- $(l, \mathcal{Z}) \rightarrow (l, (Z', \pi'))$, *where* $Z' = Z^{\uparrow} \wedge Inv(l)$, *and* $\pi'(v') = \inf\{\pi(v) + P(l) \cdot d | d \in \mathbb{R}_{\geq 0} \wedge v \in Z \wedge v' = v + d\}$.
- $(l, \mathcal{Z}) \rightarrow (l', (Z', \pi'))$, *iff* $l \xrightarrow{g,a,r} l'$, $Z' = Inv(l') \wedge \{r\}(Z \wedge g))$, $Z' \neq \emptyset$ *and* $\pi'(v') = \inf\{\pi(v) + P((l, g, a, r, l')) | v \in Z \wedge v' = [r \mapsto 0]v\}$.

The initial state is $(l_0, (Inv(l_0) \wedge Z_0, \pi_0))$ *where* $Z_0 = \{v_0\}$ *and* $\pi_0(v_0) = 0$.

Now, the above notion of priced symbolic state and associated operations, allows an abstract algorithm for computing the minimum cost of reaching a designated goal location (Figure 4.2). The algorithm starts with the initial symbolic state $(l_0, \mathcal{Z}_0)$. The minimum cost mincost(l_g) of reaching a location l_g is defined, as in Definition 3.3, to be the infimum of the cost of concrete finite executions that reach l_g. The algorithm shown in Figure 4.2 differs from the algorithm in Figure 3.8 only in the sense that we use a zone based semantics rather than the region based from Section 3.4. These algorithms can also handle reachability problems to a set of goal states. By adding an additional location, one can translate a reachability problem to a set of states to a reachability problem of a location.

The symbolic semantic and the algorithm allows us to derive the following results, similar to Theorem 3.16 and Theorem 3.18.

Theorem 4.3 *Let l be a location of a LPTA A. Then*

$$\text{mincost}(l) = \min\{\text{mincost}(\mathcal{Z}) \mid (l, \mathcal{Z}) \textit{ is reachable}\}$$

```
COST := ∞
PASSED := {}
WAITING := [(l_0, Z_0)]
while WAITING ≠ [] do
   select (l, Z) from WAITING
   if l = l_g and mincost(Z) < COST
   then COST := mincost(Z)
   if ∀(l', Z') ∈ PASSED. l ≠ l' ∨ Z' ⋢ Z
   then add (l, Z) to PASSED
        forall (l', Z') s.t. (l, Z)→(l', Z') do
           add (l', Z') to WAITING
        od
   fi
od
```

Figure 4.2: Abstract algorithm for the minimal-cost reachability problem.

Theorem 4.4 *When the algorithm in Figure 4.2 terminates, the value of* COST *equals* mincost(l_g).

Both theorems can be proven similarly to the corresponding theorems for priced regions. Theorem 4.4 ensures that the algorithm in Figure 4.2 does indeed find the minimum cost, provided that the algorithm terminates. Assuming that we can realize the necessary operations like intersection and comparison of priced zones, one can prove that the algorithm terminates, by transforming any given LPTA to its bounded counterpart, as we did in Section 3.6. Observe that each zone Z is then a finite union of regions. We can then assign to each vertex $v \in \mathbb{N}^C$ that is an element (of the closure) of Z a cost $\pi(v)$ in $\mathbb{N}$. The cost inside a zone is then defined by the cost in the nearest vertices of the grid. Dickson's lemma then ensures that we cannot have an infinite sequence of priced zones with no pair $i < j$ such that $(Z, \pi_i) \sqsubseteq (Z, \pi_j)$.

Theorem 4.5 *The algorithm in Figure 4.2 terminates for any bounded* LPTA.

Note, that we assumed that we have provided a way to realize operations on priced zones such as comparison. In the implementation of zone based time automata model checkers termination is ensured by normalizing all zones with respect to a maximum constant M [Rok93], but for LPTAs ensuring termination also depends on the representation of priced zones. In the next section we will present such a representation for the sub-class of *Uniformly Priced Timed Automata*.

4.3 Uniformly Priced Timed Automata

Definition 4.6 (Uniformly Priced Timed Automata) *A uniformly priced timed automaton (UPTA) is an LPTA where all locations have the same rate. We refer to this rate as the rate of the UPTA.*

Lemma 4.7 *Any UPTA A with positive rate can be translated into a UPTA B with rate 1 such that $mincost(l)$ in A is identical to $mincost(l)$ in B.*

Proof [sketch] Let A be a UPTA with positive rate r. Now, let B be like A except that all constants on guards and invariants are multiplied by r and set the rate of B to 1. □

Thus, in order to find the infimum cost of reaching a satisfying state in UPTA, we only need to be able to handle rate zero and rate one.

In case of rate zero, all symbolic states reachable by the symbolic semantics have very simple cost functions: The zone is mapped to the same integer, because the cost is 0 in the initial state and only modified by the increment operation. This means that a priced zone $\mathcal{Z}$ can be represented as a pair (Z, c), where Z is a zone and c an integer, s.t. $\text{cost}(v, \mathcal{Z}) = c$ when $v \in Z$ and ∞ otherwise. Delay, reset and satisfaction are easily implementable for zones using DBMs. Increment is a matter of incrementing c and a comparison $(Z_1, c_1) \sqsubseteq (Z_2, c_2)$ reduces to $Z_2 \subseteq Z_1 \wedge c_1 \leq c_2$. In the DBM-based implementation, termination is ensured by normalizing all zones with respect to a maximum constant $\max_A$.

In case of rate one, the idea is to use zones over $C \cup \{\delta\}$, where δ is an additional clock keeping track of the cost. Priced zones $\mathcal{Z}$ are then subsets of $\mathbb{R}^{C\cup\delta}_{\geq 0}$. Consequently, we have for a priced zone $\mathcal{Z}$ and $v \in \mathbb{R}^{C}_{\geq 0}$ that $\text{cost}(v, \mathcal{Z}) = \inf\{\nu(\delta) | \nu \in \mathcal{Z} \wedge \nu|_C = v\}$[1]. Delay, reset, satisfaction and computing mincost are supported directly by DBMs. Increment translates to $\mathcal{Z}[\delta \mapsto \delta + k] = \{\nu[\delta \mapsto \nu(\delta) + k] \mid \nu \in \mathcal{Z}\}$ and is also realizable using DBMs.

The only necessary operation that is not directly implementable is the comparison. Fortunately, it can be shown that comparison between priced zones can be reduced to inclusion of zones over $C \cup \{\delta\}$. For comparison between symbolic cost states notice that $\mathcal{Z}_2 \subseteq \mathcal{Z}_1$ implies $\mathcal{Z}_1 \sqsubseteq \mathcal{Z}_2$, whereas the implication in the other direction does not hold in general. The following Lemma 4.8 states that comparisons can still be reduced to set inclusion provided the zone is extended in the δ dimension.

Lemma 4.8 *Let $\mathcal{Z}^\dagger = \{\nu^\dagger | \nu \in \mathcal{Z} \wedge \nu|_C = \nu^\dagger|_C \wedge \nu(\delta) \leq \nu^\dagger(\delta)\}$. Then*

$$\mathcal{Z}_1 \sqsubseteq \mathcal{Z}_2 \Leftrightarrow \mathcal{Z}_2^\dagger \subseteq \mathcal{Z}_1^\dagger$$

[1] We define $\text{cost}(v, \mathcal{Z})$ to be ∞ if $v \notin \mathcal{Z}$

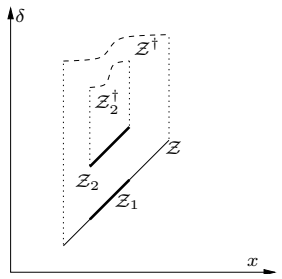


Figure 4.3: Let x be a clock and let δ be the cost. In the figure, $\mathcal{Z} \sqsubseteq \mathcal{Z}_1 \sqsubseteq \mathcal{Z}_2$, but only $\mathcal{Z}_1$ is a subset of $\mathcal{Z}$. The $()^\dagger$ operation removes the upper bound on δ, hence $\mathcal{Z}_2^\dagger \subseteq \mathcal{Z}^\dagger$.

Proof Assume $\mathcal{Z}_1 \sqsubseteq \mathcal{Z}_2$. By definition $\mathcal{Z}_1 \sqsubseteq \mathcal{Z}_2 \Leftrightarrow \forall v : \text{cost}(v, \mathcal{Z}_1) \leq \text{cost}(v, \mathcal{Z}_2)$. First, assume $\nu^\dagger \in \mathcal{Z}_2^\dagger$. Then there exist a $\nu \in \mathcal{Z}_2$ with $\nu(\delta) \leq \nu^\dagger(\delta)$ and $\nu|_C = \nu^\dagger|_C$. From $\mathcal{Z}_1 \sqsubseteq \mathcal{Z}_2$ follows $\text{cost}(\nu|_C, \mathcal{Z}_1) \leq \text{cost}(\nu|_C, \mathcal{Z}_1)$, en thus that there exists a $\nu' \in \mathcal{Z}_1$, such that $\nu'(\delta) \leq \nu(\delta) \leq \nu^\dagger(\delta)$ and $\nu|_C = \nu'|_C = \nu^\dagger|_C$. Hence, as desired, $\nu^\dagger \in \mathcal{Z}_1^\dagger$.

Assume $\mathcal{Z}_2^\dagger \subseteq \mathcal{Z}_1^\dagger$. From the definitions follows straightforward $\text{cost}(v, \mathcal{Z}) = \text{cost}(v, \mathcal{Z}^\dagger)$. Furthermore, we trivially have $\text{cost}(v, \mathcal{Z}) \leq \text{cost}(v, \mathcal{Z}')$, if $\mathcal{Z}' \subseteq \mathcal{Z}$. Hence, $\mathcal{Z}_1 \sqsubseteq \mathcal{Z}_2$ as desired. □

See Figure 4.3 for an example of the $()^\dagger$-operation. It is straightforward to implement the $()^\dagger$-operation on DBMs. A useful property of the $()^\dagger$-operation is, that its effect on zones can be obtained without implementing the operation. Suppose that $(l', \mathcal{Z}')$ is a symbolic successor of $(l, \mathcal{Z})$, due to a delay transition or discrete transition. We then have that $(l', (\mathcal{Z}')^\dagger)$ is a successor of $(l, \mathcal{Z}^\dagger)$ due to the same transition – intuitively because δ is never reset and no guards or invariants depend on δ. It is therefore sufficient to apply the $()^\dagger$ operation only once to the initial symbolic state $(l_0, \mathcal{Z}_0^\dagger)$.

Termination is ensured for the DBM-based implementation, since all clocks except for δ are normalized with respect to a maximum constant M. It is important that normalization never touches δ. With this modification, the algorithm in Figure 4.2 will essentially encounter the same states as the traditional forward state-space exploration algorithm for timed automata, except for the addition of clock δ.

4.4 Improving the State-Space Exploration

As mentioned before, a drawback of the algorithm in Figure 4.2 is, that the complete states space has to be searched. This can in most cases be improved in a

```
PASSED := {}
WAITING := [(l_0, Z_0)]
while WAITING ≠ [] do
   select (l, Z) from WAITING with smallest mincost(Z)
   if l = l_g then return mincost(Z)
   if ∀(l', Z') ∈ PASSED. l ≠ l' ∨ Z' ⋢ Z
   then add (l, Z) to PASSED
        forall (l', Z') s.t. (l, Z)→(l', Z') do
           add (l', Z') to WAITING
        od
   fi
od
```

Figure 4.4: State-space exploration algorithm using MC order.

number of ways. In analogy with Dijkstra's shortest path algorithm and the UPPAAL state-space search leads us to stop the search as soon as a goal state has been found. This is, however, based on a kind of breadth-first search which might not succeed for systems with very large state-spaces. In this case techniques inspired by branch and bound algorithms can be helpful.

4.4.1 Minimum Cost Order

The algorithm of Figure 4.2 may select, in analogy with Dijkstra's algorithm for finding the shortest path in a directed weighted graph, the symbolic state $(l, \mathcal{Z})$ from WAITING for which $\mathcal{Z}$ has the smallest minimum cost. With this choice, we may terminate the algorithm as soon as a goal state is selected from WAITING. We will refer the search order arising from this strategy as the Minimum Cost order (MC order).

Lemma 4.9 *Using the MC order, an optimal solution is found by the algorithm in Figure 4.2 when a goal state is selected from* WAITING *the first time.*

Proof When a state is taken from WAITING using the MC order, no state with lower cost are reachable. Therefore, when the first goal state is taken from WAITING no (goal) states with lower cost are reachable, so the optimal solution has been found. □

When applying the MC order, the algorithm in Figure 4.2 can be simplified as depicted in Figure 4.4. There is no need to maintain variable COST, and the algorithm stops as soon as it finds a state $(l_g, \mathcal{Z})$.

If there are more than just one symbolic state with the same minimum cost, the MC order offers no indication as to which one to explore first. As a matter of fact,

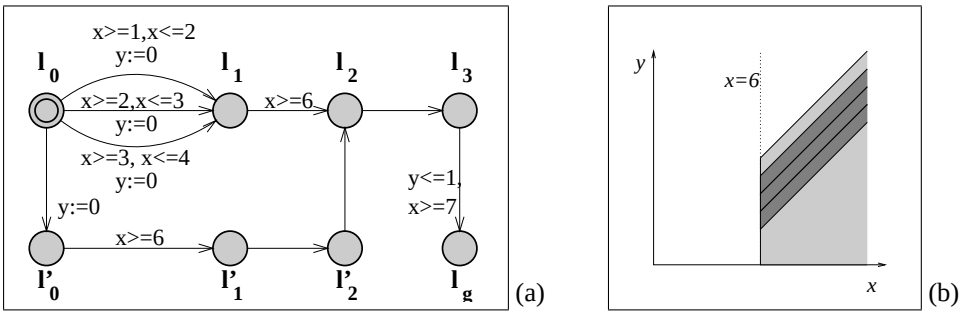


Figure 4.5: (a) This automaton illustrates that the MC search order might not be optimal in any case. We assume a cost rate of 1. Figure (b) depicts the zones that are reachable in location l_2 (and l_3). The big light grey zone is reachable in location l_2 via location l'_1, whereas the three dark grey zones are reachable only via l_1.

we cannot even be sure that a particular MC search order cannot be outperformed by a non-MC order; selecting a state from the WAITING that has not the smallest mincost may reduce the number of explored states. The MC algorithm for priced timed automata differs in this respect from the shortest-path algorithm for directed weighted graphs. The main reason is that the timed automaton discards exploration of a state based on a notion of superset, rather than equality.

Figure 4.5 illustrates this fact. The minimal cost of reaching location l_g of the automaton in Figure 4.5 (a) is 7, since clock x is never reset and hence equal to the additional clock δ. The MC order might explore first the symbolic states in locations l_0, l'_0 and l_1, and then the successors of location l_1, and finally the successors of location l'_1. Exploring in contrast the successors of l'_1 with minimal cost of 6, prior to the successors of l_1, with minimal cost of 1 to 3, reduces the number of explored states with 6; but this search order is not an MC order. The reduction is caused by the fact that the large zone in location l_2 and l_3, that contains the three smaller zones, is only reachable from location l'_1 (Figure 4.5 (b)). If we explore the large zone first, the smaller ones will not pass the test on the PASSED list, and will thus not be explored further.

Minimal-cost search orders constitutes nevertheless an important class, as stated by the following lemma:

Lemma 4.10 *There exist a MC order, such that the algorithm in Figure 4.2 explores the fewest symbolic states.*

Proof We prove this indirectly, by proving that for any search order, there exists an MC order that explores as many or less states. Note, that the minimal cost of a state never decreases if we take a transition. Given an arbitrary search order, we denote the PASSED list after termination with $\text{PASSED}_{\text{nonmc}}$. The desired MC order

is realized by the selecting states from the WAITING list according to the following precedence rules:

- Select a state with the smallest mincost.
- To solve remaining ties, select the state than was explored first by the given search order.

This search order is clearly a minimal-cost order. The second rule ensures that the predecessor of a state will be explored first. If $(l, \mathcal{Z})$ was discarded with the given non-MC order, there exists a state $(l', \mathcal{Z}') \in \text{PASSED}_{\text{nonmc}}$ with $(l', \mathcal{Z}') \sqsubseteq (l, \mathcal{Z})$. If we use the MC order that is achieved by the precedence rules, $(l, \mathcal{Z})$ will also be discarded, since $(l', \mathcal{Z}')$ will be explored before $(l, \mathcal{Z})$. The MC order will thus encounter only states that were explored by the given order, and it will discard at least as many states. But, as stated before, we may stop exploration as soon as we find a state in the goal location, and thus explore even less states. □

This proof gives a recipe on how to obtain an MC order from any given order. In the case of the example in Figure 4.5, this order imposes that we select first the states in location l_0, l'_0 and l_1, then the successors of the symbolic state in l'_1, and finally the remaining symbolic states in l_2. These will be discarded, since they are subset of a symbolic state on the PASSED list.

The results in this section are based on two implicit assumptions. We assume that we do not have any information on the minimal cost of the successors of a state; except that it does not decrease. Such information, if present, is however useful to reduce the number of explored states. Subsection 4.4.2 proposes an improved minimal-cost order. We also assumed that there is a way to define arbitrary search orders. This is in particular useful, if the state-space is that big that we cannot afford (in terms of time and memory) to wait for termination. We will propose in Subsection 4.4.3 modifications of the basic algorithm, that allows to define heuristic search orders.

4.4.2 Using Estimates of the Remaining Cost

From a given state one often has an idea about the cost remaining in order to reach a goal state. In branch-and-bound algorithms this information is used both to delete states and to search the most promising states first. Using information about the remaining cost can also decrease the number of explored states.

For a state (l, v) let $rem((l, v))$ be the minimum cost of reaching a goal state from that state. In general we cannot expect to know exactly what the remaining cost of a state is. We can instead use an estimate of the remaining cost as long as the estimate does not exceed the actual cost. For a symbolic state $(l, \mathcal{Z})$ we require

that $\text{REM}(l, \mathcal{Z})$ satisfies $\text{REM}(l, \mathcal{Z}) \leq \inf\{rem((l, v)) \mid v \in \mathcal{Z}\}$, i.e. $\text{REM}(l, \mathcal{Z})$ offers a lower bound on the remaining cost of all the states with location l and clock valuation within priced zone $\mathcal{Z}$.

Combining the minimum cost $\text{mincost}(\mathcal{Z})$ of a symbolic cost state $(l, \mathcal{Z})$ with the estimate of the remaining cost $\text{REM}(l, \mathcal{Z})$, we can base the MC order on the sum of $\text{mincost}(\mathcal{Z})$ and $\text{REM}(l, \mathcal{Z})$. Since $\text{mincost}(\mathcal{Z}) + \text{REM}(l, \mathcal{Z})$ is smaller than the actual cost of reaching a goal state, the first goal state to be explored is guaranteed to have optimal cost. We call this the MC+ order, and it is also known as Least-Lower-Bound order. In Section 4.5 we will show that even simple estimates of the remaining cost can lead to large improvements in the number of states searched to find the minimum cost of reaching a goal state.

One way to obtain a lower bound is to specify an initial estimate and annotate each transition with updates of this estimate. In this case it is the responsibility of the user to guarantee that the estimate is actually a lower bound in order to ensure that the optimal solution is not deleted. This also allows the user to apply her understanding and intuition about the system.

To obtain a lower bound of the remaining cost in an *automatic* and *efficient* manner, we suggest to replace one or more automata in the network with "more abstract" automata. The idea is that this should result in an abstract network which contains (at least) all runs of the original one, with no larger costs. Thus computing the minimum cost of reaching a goal state in the abstract network will give the desired lower bound estimate of reaching a goal state in the original network. Moreover, the abstract network should be substantially simpler to analyze than the original network making it possible to obtain the estimate efficiently.

4.4.3 Heuristics and Bounding

It is often useful to quickly obtain an initial solution and thus an upper bound on the cost. This is in particular the case the state-space is too big for the MC order to handle. As will be shown in Section 4.5, the techniques described here for altering the search order using heuristics are very useful. In addition, techniques from branch-and-bound algorithms are useful for improving the upper bound once it has been found. Applying knowledge about the goal state has proven useful in improving the state-space exploration [RE99, HLP00, Feh00a], either by changing the search order from the standard depth or breadth-first, or by pruning parts of the state-space.

To implement the MC order a suitable data-structure for WAITING would be a priority queue where the priority is the minimum cost of a symbolic cost state. We can obviously generalize this by extending a symbolic cost state with a new field, `heur`, which is the priority of the state used by the priority queue. Allowing

various ways of assigning values to `heur` combined with choosing either to first select a state with large or small priority opens for a large variety of search orders.

Annotating the model with assignments to `heur` on the transitions, is one way to allow the user to guide the search. Because of its flexibility it proves to be a very powerful way of guiding the search. The assignment works like a normal assignment to integer variables and with the same kind of expressions. We give several examples in Section 4.5.

When searching for an error state in a system a *random* search order might be useful. We have chosen to implement a *random depth-first order* which as the name suggests is a variant of a depth-first search. The only difference between this and a standard depth-first search is that before pushing all the successors of a state on to WAITING (which is implemented as a stack), the successors are randomly permuted.

Once a reachable goal state has been found, a variable COST gives an upper bound on the minimum cost of reaching a goal. If we choose to continue the search, a smaller upper bound might be obtained. During state-space exploration the cost never decreases therefore states with cost bigger than COST cannot lead to an optimal solution, and can therefore be deleted. The estimate of the remaining cost as defined in Section 4.4.2 can also be used for pruning states. Whenever $\text{mincost}(\mathcal{Z}) + \text{REM}(l, \mathcal{Z})$ is larger than the best upper bound COST no state successor of $(l, \mathcal{Z})$ can lead to a better solution than the one already found. We can therefore safely prune $(l, \mathcal{Z})$.

All of the methods described in this section have been implemented in UPPAAL. Section 4.5 reports on experiments using these new methods.

4.5 Experiments

In this section we illustrate the benefits of extending UPPAAL with heuristics and costs through several verification and optimization problems. All of the examples have previously been studied in the literature. First, however we have to define parallel composition of LPTAs.

Following the common approach to networks of timed automata, we extend LPTA to networks of LPTA by introducing a partial function $f : (Act \cup \{\iota\}) \times (Act \cup \{\iota\}) \hookrightarrow Act$, where ι is a distinguished no-action symbol.[2] We assume that f is associative and commutative. In addition, two functions $h_L, h_E : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ for combining prices of transitions and locations are introduced.

[2] We extend the edge set E such that $l \xrightarrow{tt,\iota,\emptyset,0} l$ for any location l. This allows synchronization functions to implement internal τ actions.

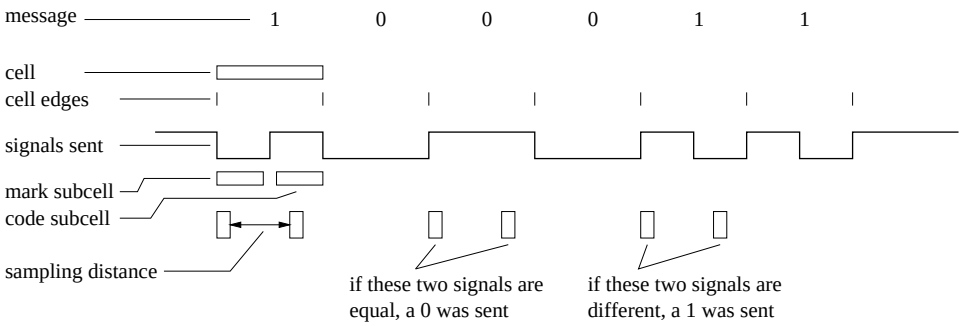


Figure 4.6: Biphase mark terminology

Definition 4.11 (Parallel Composition) *Let* $A_i = (Loc_i, l_{i,0}, E_i, Inv_i, P_i)$, $i = 1, 2$ *be two LPTA. The parallel composition is defined as* $A_1 \mid^{f}_{h_L,h_E} A_2 = (Loc_1 \times Loc_2, (l_{1,0}, l_{2,0}), E, Inv, P)$, *with* $l = (l_1, l_2)$, $Inv(l) = Inv_1(l_1) \wedge Inv_2(l_2)$, $P(l) = h_L(P_1(l_1), P_2(l_2))$, *and* $l \xrightarrow{g,a,r} l'$ *iff there exist* g_i, a_i, r_i *such that* $f(a_1, a_2) = a$, $l_i \xrightarrow{g_i,a_i,r_i} l'_i$, $g = g_1 \wedge g_2$, $r = r_1 \cup r_2$, *and* $P(l, g, a, r) = h_E(P_1(l_1, g_1, a_1, r_1, l'_1), P_2(l_2, g_2, a_2, r_2, l'_2))$.

Useful choices for h_L and h_E guaranteeing commutativity and associativity of parallel composition are summation, minimum and maximum. In the remainder we will use the sum of the prices. The first example in this section has no notion of cost at all, but it illustrates guided search.

4.5.1 The Biphase Mark Protocol

The Biphase Mark Protocol is a convention for transmitting strings of bit and clock pulses simultaneously as square waves. This protocol is widely used for communication in the ISO/OSI physical layer; for example, a version called "Manchester" is used in the Ethernet. The protocol ensures that strings of bits can be submitted and received correctly, in spite of clock drift, jitter and filtering by the channel. A formal parameterized timed automaton model of the Biphase Mark Protocol was given in [Vaa01], where also necessary and sufficient conditions on the correctness for a parametric model were derived. We will use the corresponding UPPAAL models to investigate the benefits of heuristics in pure reachability analysis.

The model assumes that sender and receiver have both their own clock with drift and jitter. The sender encodes each bit in a *cell* of length c clock cycles (see Figure 4.6). At the beginning of each cell, the sender toggles the voltage. The sender then waits for m clock cycles, where m stands for the *mark* subcell. If the sender has to encode a "0", the voltage is held constant throughout the whole

Table 4.7: Results for nine erroneous instances of the Biphase Mark Protocol. Numbers of state explored before reaching an error state

	nondetection mark subcell			sampling early			sampling late		
	(16,3,11)	(18,3,10)	(32,3,23)	(16,9,11)	(18,6,10)	(32,18,23)	(15,8,11)	(17,5,10)	(31,16,23)
breadth first	1931	2582	4049	990	4701	2561	1230	1709	3035
`in==1` heuristic	1153	1431	2333	632	1945	1586	725	1039	1763

cell. If it encodes a "1" it will toggle the voltage at the end of the mark subcell. The signal is unreliable during a small interval after the sender generates an edge. Reading it during this interval may produce any value.

The receiver waits for an edge that signals the arrival of a cell. Upon detecting an edge, the receiver waits for a fixed number of clock cycles, the *sampling distance* s, and samples the signal. We adopt the notation $bpm(c, m, s)$ for instances of the protocol with cell size c, mark size m and sampling distance s.

There are three kind of errors that may occur in an incorrect configuration. Firstly, the receiver may not detect the mark subcell. Secondly, the receiver may sample too early, before or right after the sender left the mark subcell. Finally, the receiver may also sample too late, i.e. the sender has already started to transmit the next cell. The first two errors can only occur if there is an edge after the mark subcell. This is only the case if input 1 is offered to the coder. The third error seems to be independent of the offered input.

Since two of the three errors occur only if input 1 is offered to the coder, and the third error can occur in any case, it seems worthwhile to choose a heuristic that searches for states with input 1 first, rather than exploring state-space for both possible inputs concurrently. Standard breadth-first search can be obtained by adding the assignment `heur := heur-1` to each transition and selecting the symbolic state with the highest value from WAITING. This can be done by adding a global assignment to the model. Giving very low priority to the part of the state-space where a 0 has been send we will obtain the desired search order. The choice of what to send is made in one place in the model of the sender. We add on the transition that models sending a 0 the assignment `heur := heur-1000` which will give this state and all its successors very low priority, and therefore these will be explored last. In this way we do not leave out any part of the state-space, but first search the part that we consider to be the most interesting. We apply this heuristic to erroneous modifications of the (correct) instances $\text{BPM}(16, 6, 11)$, $\text{BPM}(18, 5, 10)$ and $\text{BPM}(32, 16, 23)$. Table 4.7 gives the results.

It turns out that only about 60% of the complete state-space size is explored.

Table 4.8: Computational results for the bridge problem by Ruys and Brinksma.

	Initial Solution		Optimal Solution		With est. remainder	
	states	cost	states	cost	states	cost
BF	4491	65	4539	60	4493	60
DF	169	685	25780	60	5081	60
MC	1536	60	1536	60	N/A	N/A
MC+	404	60	404	60	N/A	N/A

The corresponding diagnostic traces show that the errors were found within the first cell or at the very beginning of the second cell, thus at a stage were only one bit was sent and received. Exploring only the part of the state-space with input 1 in the first cell, saves about 40 % of the state-space. This is less than a half, since for input "1" there is more activity in the protocol. An exception on this rule is the fifth instance BPM$(18, 6, 10)$, which produces an error after one and a half cell, and shows consequently a larger reduction when verified with the heuristic.

4.5.2 The Bridge Problem

The following problem was proposed by Ruys and Brinksma [RB98]. A timed automaton model of this problem is included in the standard distribution of UPPAAL[3].

Four persons want to cross a bridge in the dark. The bridge is damaged and can only carry two persons at the same time. To cross the bridge safely in the darkness, a torch must be carried along. The group has only one torch to share. Due to different physical abilities, the four cross the bridge at different speeds. The time they need per person is (one-way) 25, 20, 10 and 5 minutes, respectively. The problem is to find a schedule such that all four cross the bridge within a given time. This can be done with standard UPPAAL. With the proposed extension, it is also possible to find the fastest time for the four to cross the bridge, and a schedule achieving this.

We compare four different search orders: Breadth-First (BF), Depth-First (DF), Minimum Cost (MC) and an improved Minimum Cost (MC+). In this example we choose the lower bound on the remaining cost, REM$(\mathcal{Z})$, to be the time needed by the slowest person, who is still on the "wrong" side of the bridge.

For the different search orders, Table 4.8 shows the number of states explored to find the initial and the optimal time, and the values of the times. It can be seen that BF explores 4491 states to find an initial schedule and 4539 to find the optimal one. This number is reduced to 4493 explored states if we prune the state-space based on the estimated remaining cost (third column). Thus, in this case only

[3] The distribution can be obtained at `http://www.uppaal.com`.

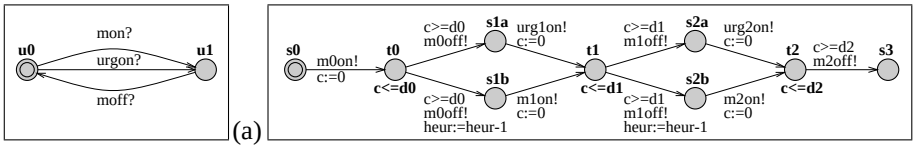


Figure 4.9: Timed automata models of a machine (a) and a job (b), that exploits *urgency*. See Figure 2.1 on page 24 for the model without urgent transitions.

two additional states are explored after the initial solution is found, all others are pruned. DF finds an initial solution (with high costs) quickly, but explores 25779 states to find an optimal schedule, which is much more than the other heuristics needed. This is most likely caused by the fact that DF search encounters many small and incomparable zones. In any case, it appears that the depth-first strategy always explores many more states than any other heuristic.

Searching with the MC order does indeed improve the results, compared to BF and DF. It is however outperformed by the MC+ heuristic that explores only 404 states to find a optimal schedule. Note that pruning based on the estimate of the remaining cost does not apply to MC and MC+ order, since the first explored goal state has the optimal value.

Without costs and heuristics, UPPAAL can only show whether a schedule exists. Extended UPPAAL finds the optimal schedule and explores with the MC+ heuristic only about 10% of the states that are needed to find a initial solution with the breadth-first heuristic.

4.5.3 Job Shop Scheduling

We apply UPPAAL to 25 of the smaller Lawrence Job Shop problems.[4] Our models are based on the timed automata models presented in Chapter 2. Recall that each operation on machine `X` is modeled by two transitions; one labeled `mXon` to model the start and another labeled `mXoff` to model the completion of an operation. If we declare the on-transition urgent, we minimizes the time a job can spend in between two operations. As a consequence we get, as reported in Section 2.6, fairly good initial solutions, but we can no longer guarantee that the optimal solution is part of the state-space.

The modified model in Figure 4.9 shows how to introduce urgency such that the optimal solution can still be computed. Two different transition are used to mark the beginning of an operation. The transitions labeled `urgXon` are urgent. If the automaton in Figure 4.9 (b) is in either location `S1a` or `S2a`, and the outgoing

[4] These and other benchmark problems for job shop scheduling can be found on `ftp://ftp.caam.rice.edu/pub/people/applegate/jobshop/`.

transition is enabled, time may not advance. The next operation will start as soon as the corresponding machine is available. If the automaton is in contrast in location `S1b` or `S2b`, the automaton may delay. In order to favor the urgent transition we decorate the off-transitions with assignments to the priority field `heur`. We assume that we select symbolic states with the highest priority. Since we expect that most delays between operations do not contribute to a good solution, we decrease the priority `heur`, whenever the automaton takes a transition to a location with a non-urgent outgoing on-transition.

The invariants in the timed automaton model are not necessary for the correctness of the model. Leaving out however results in poorer results. The same holds for an alternative model that omits the automata that model the machines. Machines are represented in this alternative approach by clocks. If a machine was untouched for at least the duration of the operation, the job automaton may take a corresponding transition. It then resets the clock. The timed automaton model job will only have m transition, for a job with m operations, but also $n + m$ clocks rather than n. As a consequence, this model does not lead an improvement of the computational results.

In order to estimate the lower bound on the remaining cost, we calculate for each job and each machine the duration of the remaining operations. We obtained this bounds in two ways. Firstly, we define lower bounds analogously to invariants, i.e. we put a lower bound that holds as long as we are in a certain location. For example, as long as the automaton in Figure 4.9 stays in location `S2b`, we know that the remaining cost is at least `d2`. If we leave the location this lower bound decreases as time advances, unless we have another lower bound.

Secondly, certain transitions may imply a lower bound on the remaining cost. After the transition has been taken, this lower bound may decrease as the cost increases, as long as this does not violate another lower bound that depends on locations. If we take for example transition `m1on` in Figure 4.9 (b), we know for sure that it will take at least another `d1` + `d2` time units to reach location `S3`. This kind of lower bounds can be considered as counterpart of guards on transitions. If the transition is taken, we know that the guard holds. Time may delay afterwards. Similarly, the lower bound should hold if the transition is taken, but it may decrease as time passes. These estimates may be seen as obtained by abstracting the model to one automaton as described in Section 4.4.2. The final estimate of the remaining cost is then estimated to be the maximum of bounds obtained for the individual jobs and machines.

Table 4.10 shows results obtained for the search orders BF, MC, MC+, DF, Random DF, and a combined heuristic. The latter is based on depth-first but takes also into account the remaining operation times and the lower bound on the cost, via a weighted sum which is assigned to the priority field of the symbolic states.

Table 4.10: Results for the 25 job shop problems with 5 machines and 10 jobs (la1-la5), 15 jobs (la6-la10) and 20 jobs (la11-la15), and 10 problems with 10 machines, 10 jobs (la16-20) and 15 jobs (la21-25). The table shows the best solution found by different search orders within 60 seconds cpu time on a Pentium II 300 MHz. If the search terminated also the number of explored states is given. The last row gives the makespan of an optimal solution.

problem	BF		MC		MC+		DF		RDF		comb. heur.		minimal
instance	cost	states	cost	states	cost	states	cost	states	cost	states	cost	states	makespan
la01	-	-	-	-	-	-	2466	-	842	-	666	292	666
la02	-	-	-	-	-	-	2360	-	806	-	672	-	655
la03	-	-	-	-	-	-	2094	-	769	-	626	-	597
la04	-	-	-	-	-	-	2212	-	783	-	639	-	590
la05	-	-	-	-	593	9791	1955	-	696	-	593	284	593
la06	-	-	-	-	-	-	3656	-	1076	-	926	480	926
la07	-	-	-	-	-	-	3410	-	1113	-	890	-	890
la08	-	-	-	-	-	-	3520	-	1009	-	863	400	863
la09	-	-	-	-	-	-	3984	-	1154	-	951	425	951
la10	-	-	-	-	-	-	3681	-	1063	-	958	454	958
la11	-	-	-	-	-	-	4974	-	1303	-	1222	642	1222
la12	-	-	-	-	-	-	4557	-	1271	-	1039	633	1039
la13	-	-	-	-	-	-	4846	-	1227	-	1150	662	1150
la14	-	-	-	-	1292	10653	5145	-	1377	-	1292	688	1292
la15	-	-	-	-	-	-	5264	-	1459	-	1289	-	1207
la16	-	-	-	-	-	-	4849	-	1298	-	1022	-	945
la17	-	-	-	-	-	-	4299	-	938	-	786	-	784
la18	-	-	-	-	-	-	4763	-	1034	-	922	-	848
la19	-	-	-	-	-	-	4566	-	1140	-	904	-	842
la20	-	-	-	-	-	-	5056	-	1378	-	964	-	902
la21	-	-	-	-	-	-	7608	-	1326	-	1149	-	(1040,1053)
la22	-	-	-	-	-	-	6920	-	1413	-	1047	-	927
la23	-	-	-	-	-	-	7676	-	1357	-	1075	-	1032
la24	-	-	-	-	-	-	7237	-	1346	-	1061	-	935
la25	-	-	-	-	-	-	7141	-	1290	-	1070	-	977

The results show BF and MC order cannot complete a single instance in 60 seconds, but even when allowed to spend more than 30 minutes using more than 2Gb of memory no solution was found. It is important to notice that the combined heuristic used includes a clever choice between states with the same values of cost plus remaining cost. This is the reason it is able to outperform the MC+ order which is only able to find solution to two instances within the time limit of 60 seconds.

As can be seen from the table UPPAAL is handling the first 15 examples quite well; it finds the optimal solution in 11 cases and it shows that it is optimal in 10 cases. This is much more than without the added guiding features. For the 10 largest problems (la16 to la25) with 10 machines we did not find optimal solutions though in some cases we were close to the optimal solution. Since branch-and-bound algorithms generally do not scale too well when the number of machines

and jobs increase, this is not surprising. The branch-and-bound algorithm presented in [AC91], who solves about 10 out of the first 15 problems in the same setting, faces the same problem. Note that the results of this algorithm depend sensitively on the choice of an initial upper bound. Also the algorithm used in [BJS95], which combines a good heuristic with an efficient branch-and-bound algorithm and thus solves all of these 15 instances, does not find solutions for most of the larger instances with 15 jobs and 10 machines or larger.

Recent work by Abdeddaïm and Maler [AM01] shows that it is possible to tailor the model checking algorithm for timed automata to job shop problems. For this class for problems it is for example sufficient to consider only the lower bounds of a zone; this in contrast with our approach that removes only the upper bound on clock δ. As a consequence they can will prune more symbolic states during the exploration of the state-space. They presented results, obtained with the model checker KRONOS, for 12 instances of the job shop problem, of which some are also instances in Table 4.10. These results confirm, even though they were obtained without a limit on the cpu time, that our results can be improved, if one moves towards algorithms that are more dedicated to the problem domain.

4.5.4 The Sidmar Steel Plant

The model of the Sidmar Steel Plant as presented in Section 2.5 contained already some modifications to ease the state-space exploration. It assigns for example to each ladle a number to reduces the symmetry. But still, its state-space is very large since at each point in time many different possibilities are enabled. Consequently, depth-first search needed about 50 minutes cpu time on a Pentium III 500MHz to find a schedule for a plant with 5 batches and one crane.

Priorities can be used to influence the search order of the state-space, and thus to improve the results. Based on a depth-first strategy, we reward transitions that are likely to serve in reaching the goal, whereas transitions that may spoil a partial solution result in lower priorities. For instance, when a batch of iron is being treated by a machine, it pays off to reward other scheduling activities that take place in the meanwhile, rather than wait for the treatment to finish.

To estimate the remaining time, we determined for each recipe the remaining time until completion. An additional lower bound is determined by the casting machine. It takes at least 20 time units to empty a ladle. Additionally, the first ladle has to wait before it enters the casting machine, and the last one has to wait after it leaves the machine. We can add to this bound also the time it takes to transport the last ladle to the storage place. The overall estimate is then the maximum of the individual bounds.

To compute the schedule in Figure 2.10, with makespan 245, UPPAAL explored

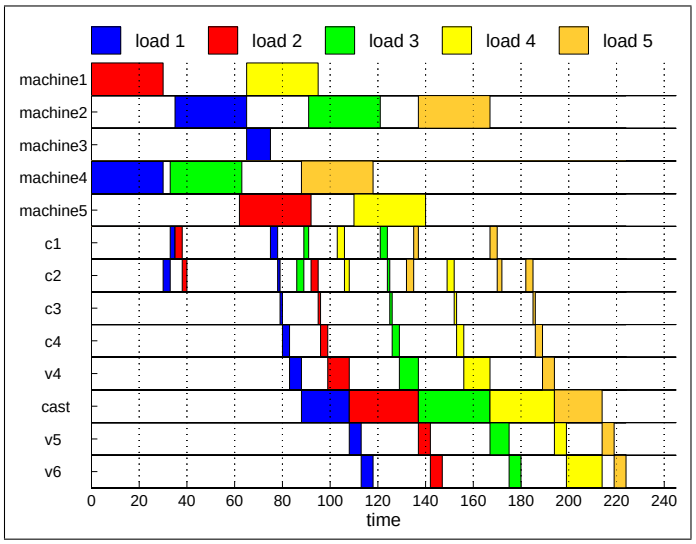


Figure 4.11: Initial schedule found by UPPAAL for the model of Sidmar steel plant with heuristics

525286 states with depth-first search. With heuristics extended UPPAAL found an initial solution with makespan 224 after exploration of 1880 states, which is just about 0.35% of the states unguided UPPAAL needs to find an initial solution. Within one minute cpu time on a Pentium III 500 MHz UPPAAL was able to compute the schedule in Figure 4.11 with makespan 216. UPPAAL explored 4827 states to find this schedule, which took 5.3 seconds cpu time, and it was not able to improve this result in the remaining time. This schedule is probably not optimal, since load 1 is first treated on machine #4 and then on machine #2. Load 2, in contrast, receives its first treatment on machine #1 and its second one treatment on machine #5. Since machine #1 and machine #4 are the same, machine #1 could treat load 1 as well, and machine #4 load 2. This would save two transports by the crane, and could probably lead to a better schedule.

The model with heuristic allows also to compute schedule for more batches. An initial schedule for a model with 10 batches and both cranes required exploration of 2334 states and has a makespan of 375. The best solution UPPAAL finds within one minute, has a makespan of 359, and took 9201 explored states. In [HLP00] schedules for up to 60 ladles were produced also using UPPAAL. To do so, additional constraints were included that reduce the size of the state-space drastically, but also prune possibly sensible behavior. A similar reduced model was used by Stobbe in [Sto00], who uses constraint programming to schedule 30 ladles. All these works consider only ladles with the same quality of steel and the initial solutions were not

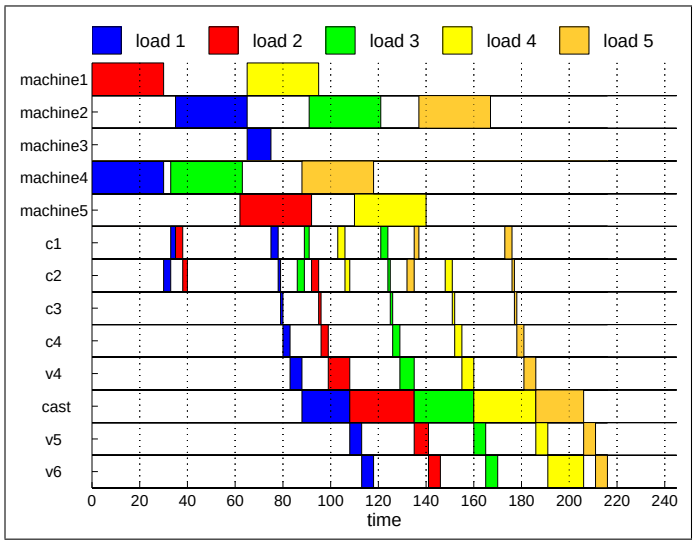


Figure 4.12: The best schedule UPPAAL found within one minute for the model with heuristics.

improved.

4.5.5 The Experimental Batch Plant

Another case study from the VHS project is the experimental batch plant, originally designed at the University of Dortmund for student exercises. Figure 4.13 depicts the piping and instrumentation diagram of the plant. It produces batches of diluted salt solution from concentrated salt solution (in container B1) and water (in container B2). These ingredients are mixed in container B3 to obtain the diluted solution, which is subsequently transported to container B4 and then further on to B5. In container B5 an evaporation process is started. The evaporated water goes via a condenser to container B6, where it is cooled and pumped back to B2. The remaining hot, concentrated salt solution in B5 is transported to B7, cooled down and then pumped back to B1.

The plant is controlled by a *Programmable Logic Controller* (PLC). PLCs are special purpose computers designed for control tasks. The most significant difference with usual computers is that a program on a PLC runs in a permanent loop, the so called *scan cycle*. In each scan cycle the program in the PLC is executed once, where the program execution may depend on variable values stored in the memory.

The length of a scan cycle is in the range of milliseconds, depending on the

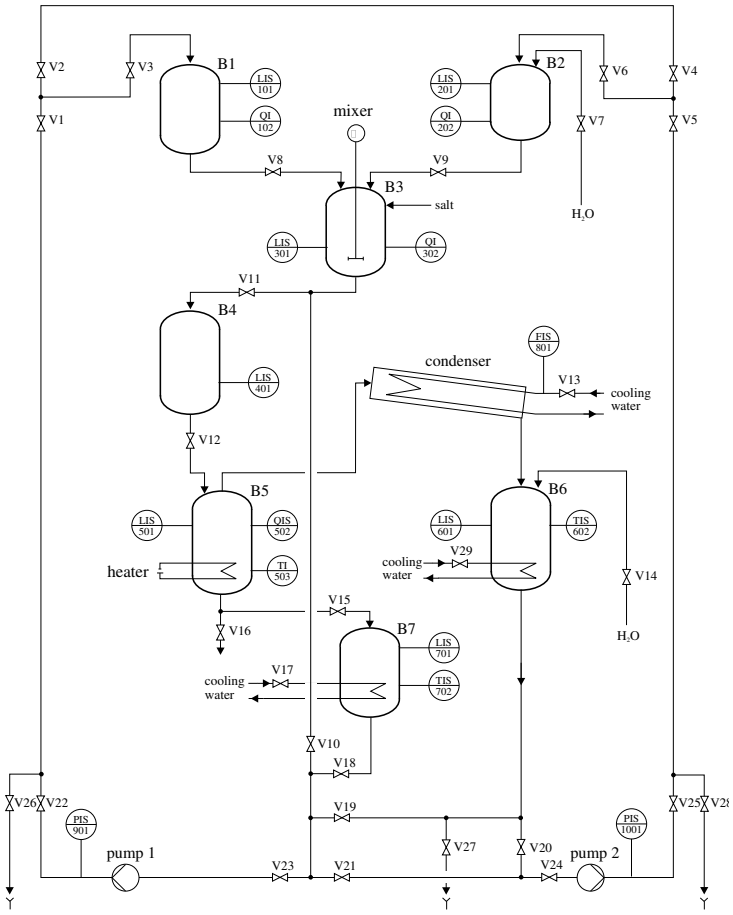


Figure 4.13: The P/I-diagram of the batch plant

length of the program. Furthermore, a part of each scan cycle is dedicated to data exchange with the environment: a PLC has *input points* connected via an interface with a dedicated *input area* of its memory, and the *output area* of the memory is connected via an interface with the *output points* of the PLC. On the input points the PLC receives data from sensors, on the output points the PLC sends data to actuators. Finally, there are some activities of the operating system (such as self checks and evaluating watchdogs) that take place in a scan cycle. The operation system itself is small and stable, which is prerequisite for reliable real-time control.

The production process can be dissected into a number of transport processes, such as transport of salt solution from container B1 to B3. All possible transport processes, the evaporation process and two cooling processes lead to 12 parallel

```
process P13{
clock x12; state S0, S1{x12<=32}, S2, S3;
commit S2, S3; init S0;
trans S0 -> S1{guard Pump1==0,V8==1, V9==0, V11==0, Mixer==1;
               sync urgon?; assign x12:=0; },
S1 -> S2 {guard x12==32, B1==1, B3==0; assign B1:=0, B3:=2; },
S1 -> S2 {guard x12==32, B1==2, B3==0; assign B1:=1, B3:=2; },
S1 -> S2 {guard x12==32, B1==1, B3==1; assign B1:=0, B3:=3; },
S1 -> S2 {guard x12==32, B1==2, B3==1; assign B1:=1, B3:=3; },
S2 -> S3 {sync compute!; },
S3 -> S0 {sync compute!; }; }
```

Figure 4.14: The UPPAAL model of transfer between container B1 and B3, which has a duration of 32 time units.

processes. The activities in each process are simply to open some valves, switch on a mixer, pump or heater, and when the process finished, close and switch off everything again. Each process starts its activities if its *activation conditions* are fulfilled, and is in a wait state otherwise. An active process remains active until its postconditions are fulfilled; it then gets back in the waiting state. The control program then decides which process to enable, in order to establish that under ideal circumstances, without leakage and unexpected evaporation, new batches will always be produced.

The translation from the plant and control to a UPPAAL model is straightforward. The plant is modeled as a parallel composition of 12 automata, where each represents one of the processes. Each of these plant automata is equipped with a clock that measures the duration of the process after activation. An example for the transport of concentrated salt solution from container B1 to container B3 is contained in Figure 4.14. The process is initially in state S0 when it is passive. The process starts as soon as the control opens and closes the appropriate valves, switches on the mixer and the appropriate pump. The process is forced to start without delay by a synchronization on an *urgent channel*, called *urgon*. After the time has passed by that the process takes, the container volumes change depending on their previous values. The different possible contents of a container are encoded by integers from 0 to 4, depending on the process.

We assume that the execution of the PLC control program can be considered to be instantaneously, compared to the time scale on which the transport processes take place. Technically, this can be translated to *committed locations* as provided by UPPAAL, that have to be left without time delay or other interleaving transitions. All locations of the control are modeled as *committed locations*, apart from the first

```
A1 -> A2{assign bonus:=bonus+(T11==1?1:0);},
A1 -> A2{guard T11==1; assign T11:=0;},
A2 -> A3{assign bonus:=bonus+(T12==1 ?1:0);},
A2 -> A3{guard T12==1; assign T12:=0;},
```

Figure 4.15: Part of the model of the control program. The transition decide whether to disable a process setting a corresponding variables to 0, if it was 1 before. The heuristic rewards to not disable processes that are enabled , to increase concurrency.

one which represents the idle state of the control.

The moments of control execution are restricted in the UPPAAL model to those points of time when the conditions in the plant change, because these are the only moments when the control changes its state. More precisely, it is the case that each change in the state of the plant requires two control program executions: one to finish some process (close valves, etc.), and one to start up new ones. In general, starting up new processes could be delayed for some time. However, in the case here, it holds for optimal schedules that if a process starts, it starts at the moment that some other process finished.

The control automaton itself consists of two sequential parts. In the first part the activation conditions for processes are evaluated, including a nondeterministic choice of a subset of the processes that may be activated. This nondeterministic step is prerequisite for finding optimal schedules. In the last part the control sets the actuator variables for valves, pumps, heater and mixer.

The cost rate equals 1, since the cost is identical to the time elapsed. We added a global assignment to the priority variable `heur` that is calculated according to the expression `1*bonus+100*depth-cost`. The variable `bonus` is used to reward selecting larger rather than smaller subsets of enabled start events, as is shown in Figure 4.15. This heuristic directs the exploration such that the controller tries first to start all permissible plant processes. The bonus is made extra rewarding for the selection of the evaporation process, which should be in (almost) continuous use for an optimal exploitation of the plant resources.

The variable `depth` is used to reward depth-first over breadth-first search. This, because a good solution for the batch plant should show some progress, i.e. it should start processes that are necessary to produce a batch. For example, we increase variable `depth` if transition `S0 -> S1` is taken in the automaton in Figure 4.14 to reward the start of process `P13`. Transitions that start other processes and transitions of the controller that enable processes to start are rewarded similarly.

The UPPAAL model of the experimental batch plant is complementary to work that was initially presented in [BM00]. Brinksma and Mader used the non-real-time model checker Spin to verify the correctness of the control program and to derive

optimal schedules for a Promela model of the plant. The intention of this approach was to see how much could be achieved here using the standard model checking environment of SPIN/Promela [Hol97]. They handled the relevant real-time properties of the PLC controller using a time-abstraction technique; for the scheduling we implemented in Promela a so-called *variable time advance procedure* [She99]. This approach aims to avoid an unnecessary blow-up of the state-space due to irrelevant points in time, i.e. times at which nothing interesting can happen. To do so it calculates at each occurrence of an event the point in time at which the *next* event will occur, and then jumping to that point in time.

For this case study these techniques proved sufficient to verify the design of the controller and derive (time-)optimal schedules with reasonable time and space requirements. One of the conclusions of the initial experiments as reported in [BM00] was that ". . . it would be useful to be able to influence the search strategy of the model checker more directly and guide the search first into those parts . . . where counterexamples are likely to be found." Since the publication of [BM00] the prototype implementation of cost-optimal UPPAAL has become available that employs a guided evaluation strategy for state-space exploration. This motivated us to carry out the optimization part of the case study again with UPPAAL.

The translation of the plant and the control program to a Promela model, or UPPAAL model respectively, is straightforward, and both models share in many points the same philosophy. The main differences of the Promela model and the UPPAAL model are of course the use of time, which is built-in in UPPAAL, and the execution of the control program. In the Promela model the latter is restricted by fairness conditions, in the UPPAAL model we restricted it explicitly, as mentioned before, to those points in time when the conditions in the plant change.

The Promela model, as presented in [BMF02], contains modifications to guide the verification tool SPIN. The model checker has for example to face like UPPAAL the exponential blow-up of the state-space that is caused by the fact that the controller may decide to disable an arbitrary number of processes. To control this phenomenon a global system parameter is introduced that specifies the maximal number of events that may be postponed by the controller. It turns out that none the obtained results required this parameter to be bigger than 2.

Spin obtained optimal schedules for instances of the model with 1 to 7 initial loads. For initial experiments we put an upper bound of 5000 time units on the makespan. For each initial load Spin needed two or three runs to determine the maximal number of batches for which counterexamples could be produced in a very short time (in the order of seconds system time). It turned out that all counterexamples contained schedules that rapidly (i.e. within 600 time units) converges to a repeating pattern with a fixed duration.

For comparison we instantiated the UPPAAL model to the same initial load and

		heuristic search			depth-first search		
load	batches	makespan	period	states	makespan	period	states
1	10	3476	380	14063	4178	458	17547
2	25	4968	206	35952	11048	458	45781
3	25	4732	206	34584	10992	458	45385
4	25	4774	206	35344	11288	458	46181
5	20	3947	206	28808	9294	458	37716
6	20	4120	206	29568	9294	458	37716
7	10	3320	346	14377	4418	458	17957

Table 4.16: This table shows the first solution found by cost-optimal UPPAAL, either with heuristic search or depth-first.

to the same number of batches as the Promela models. Table 4.16 presents the results that were obtained with UPPAAL. The column *load* indicates the number of batches with which the plant is initialized, *batches* the number of batches produced in that trace and *period* the period of the periodic behavior in time units. Clearly, UPPAAL explores with the heuristic search order less states than depth-first does, and finds even a better solution than depth-first search does. As a matter of fact, the period of 458 that was found by depth-first, is the worst possible period, since this is the sum of the durations of all transport processes.

For 1 to 6 initial loads, UPPAAL finds with the heuristic initial solutions that have exactly the same period and same makespan as the best solutions found by Spin. The initial solution for 7 loads converges to a schedule with period 346, whereas Spin is able to find a schedule that converges to a period of 314. UPPAAL was able to improve on the initial solution for 7 loads, with a schedule with a makespan of 3288. Backtracking yielded a schedule that produces the last batch within 314 time units. To do so UPPAAL explores 74240 states, which takes 16.5 seconds on a Pentium III 500MHz.

In contrast to Spin it was not necessary to define an initial upper bound on the makespan to produce schedules with UPPAAL. As a matter of fact UPPAAL can produce schedules for any reasonable number of batches, as the number of explored states grows nearly linearly with an increasing number of batches. For example, to find an initial schedule for load 4 with n batches with $n \geq 4$ UPPAAL explores $3424 + (n-4) \cdot 1520$ states. This equality holds up to 100 batches, in which case UPPAAL explores 149344 states, in 33.2 seconds, to find an initial solution with a makespan of 20224.

Six of the computed periods can be easily shown to be optimal. For a plant with an initial load of 1 this can be readily checked by hand by moving a single batch through the plant and measuring the total duration of the critical branches of

the path. The schedules for 2 to 6 initial loads can be shown to be optimal, since heating container B5 clearly dominates the time consumption during the production of batches. Since filling B5 (33 time units), heating it (147 time units), and emptying B5 (26 time units) must be part of every production cycle, the average production time of a batch must be greater or equal to 33+147+26=206 time units. This makes the schedules for 2 to 6 initial loads optimal schedules as well.

Compared to Spin, cost-optimal UPPAAL offers a more convenient interface for handling guided state-space explorations. For Spin the latter can only be done indirectly by repeated verification runs with different parameter settings under control of the user. In cost-optimal UPPAAL the user can control everything directly by defining heuristic functions, so that multiple runs may be less needed, although some additional experiments is required to fine-tune the heuristic. In view of the very reasonable performance of Spin, however, developing a Uppaal model may not pay off if a good Promela model is available. Given this situation it can be concluded that it would be very interesting to look also into the possible extension of Spin with cost-guided state-space exploration features.

Another approach to the optimal scheduling for the VHS case study 1 is reported in [NY99]. Here the problem is analyzed using the tools OpenKronos and SMI. It is difficult to compare the results of this approach directly with ours, as they include also the production of the initial loads into their schedules, which we just assume to be present. The more general findings seem to be consistent with ours. OpenKronos could be used successfully to produce optimal schedules for loads of up to 3 batches before falling victim to the state explosion problem. The symbolic model checker SMI produced results 6 batches and more, with a computation time of approximately 17 minutes per batch.

4.6 Conclusion

On the preceding pages, we have introduced (1) a priced zone based symbolic semantics for the class of linearly priced timed automata; (2) an efficient, zone based implementation of priced zones for the class of uniformly priced timed automata; (3) a number of techniques to reduce the number of explored states; and (4) experimental evidence that these techniques can lead to dramatic reductions in the number of explored states. In addition, we have shown that it is possible to quickly obtain upper bounds on the minimum cost of reaching a goal state by manually guiding the exploration algorithm using priorities.

5

Efficient Minimal-Cost Reachability for Linearly Priced Timed Automata

5.1 Introduction

Although ensuring computability, the region construction introduced in Chapter 3 is known to be very inefficient. Chapter 4 introduced *priced zones* and provided an efficient implementation via *Difference Bound Matrices* [Dil89] for the restricted class of *Uniformly Priced Timed Automata*. The central contribution of this chapter is the extension of this concept to that of *linearly priced zones*, which are attributed with an (affine) linear function of clock valuations that defines the cost of reaching a valuation in the zone. We show that the entire machinery for symbolic reachability in terms of zones can be lifted to cost-optimal reachability for linearly priced zones.

It turns out that some of the operations on linearly priced zones force us to split them into parts with different price attributes, giving rise to the notion *facets* of a zone. Consider for example the zone depicted in Figure 3.7.vii and its delay successor in Figure 3.7.viii. While the first is effectively representable as linearly priced zone, the latter does not allow to define one affine linear function on the whole zone.

The suitability of the LPTA model for scheduling problems was already illustrated in the previous chapter, using the more restricted *Uniformly Priced Timed Automata* (UPTA) model. This model was used to consider traces for the time-optimal scheduling of a steel plant, an experimental batch plant and a number of job shop problems. The greater expressivity of LPTAs also supports other measures of cost, like idle time, weighted idle time, mean completion time, earliness,

---

This chapter is based on the publication:

[LBB+01] K.G. Larsen, G. Behrmann, E. Brinksma, A. Fehnker, T.S. Hune, P. Petterson and J.M.T. Romijn. *As Cheap as Possible: Efficient Cost-Optimal Reachability for Priced Timed Automata* CAV'01.

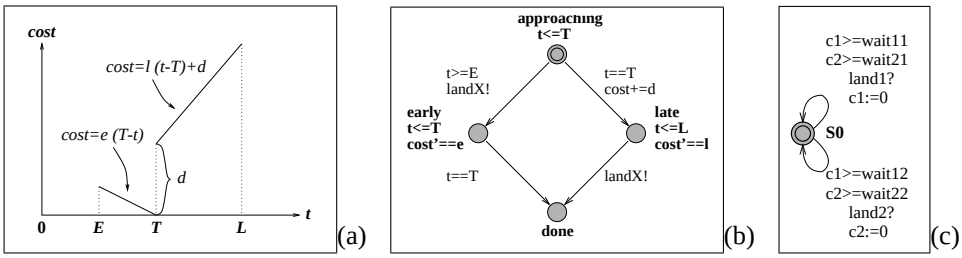


Figure 5.1: Figure (a) depicts the cost of landing a plane at time t. Figure (b) shows an LPTA modeling the landing costs. Figure (c) shows an LPTA model of the runway.

number of tardy jobs, and tardiness. Note, that in the case of job shop problems, time optimal schedules are also optimal with respect to idle time and weighted idle time. We take an aircraft landing problem [BKA00], that combines earliness with tardiness, as the application example for his chapter.

Example [Aircraft Landing Problem] The problem is to schedule a number of aircrafts safely and efficiently. For each aircraft there is a maximum speed and a most fuel efficient speed which determine an earliest and latest time the plane can land. In this interval, there is a preferred landing time, called target time, at which the plane lands with minimal cost. The target time T and the interval $[E, L]$ are shown in Figure 5.1(a). For each time unit the actual landing time deviates from the target time, the cost increases with rate e for early landings and rate l for late landings. In addition there is a fixed cost d associated with late landings.

In Figure 5.1(b) the cost of landing an aircraft is modeled as an LPTA. The automaton starts in the initial location `approaching` and lands at the moment one of the two transitions labeled `landX!` is taken. In case the plane lands too early it enters location `early` in which it delays exactly $T - t$ time units. The cost rate in this location is `e`, the cost of reaching location `done` from this location is therefore $e\,(T - t)$. In case the plane is late, the automaton takes first the transition guarded `t==T` to location `late`. The cost in location late increases with rate l. On transition `landX!` from location `late` to location `done` at time t, the cost is $l(t - T)$. Invariants ensure that the automaton always ends in location `done`, after at most L time units.

Figure 5.1(c) models a runway ensuring that two consecutive landings takes place with a minimum separation time, under the assumption that there are two types of planes. For each plane we include an LPTA as depicted in Figure 5.1(b) and for each runway an automaton as depicted in 5.1(c). We also include a dummy automaton (that is not depicted), to ensure that t is initially zero and that all transitions of the runway automata are enabled. Recall, that we defined composition of

LPTAs in Definition 4.11. We assume that the cost of delaying in the network is the sum of the cost of delaying in the individual automata. A further discussion of this example can be found in Section 5.4. □

The structure of the rest of this chapter is as follows. Section 5.2 contains the definition of the central concept of linearly priced zones. The operations that we need on linearly priced zones and facets are provided in Section 5.3. The implementation of the algorithm, and the results for the aircraft landing and other examples are reported in Section 5.4. Our conclusions, finally, are presented in Section 5.5.

5.2 Linearly Priced Zones

Typically, reachability of a timed automaton, is decided using symbolic states represented by pairs of the form (l, Z), where l is a location and Z is a zone. The framework given in Section 4.2 for symbolic computation of minimum-cost reachability extends the zone-based representation of symbolic states, and assigns costs to individual states. But, the notion of priced zones, as presented in that section, is too general to derive an immediate efficient representation. For this, we introduce the following notion of a *linearly priced* zone.

For simplicity we do not deal with strict inequalities in guards and invariants in this chapter. Therefore, $\mathcal{B}(C)$ is the set of formulas that are conjunctions of atomic constraints of the form $x \sim n$ and $x - y \sim m$ for $x, y \in C$, $\sim \in \{\leq, =, \geq\}$, with $n \in \mathbb{N}$ and $m \in \mathbb{Z}$. The *offset*, $\mathbf{0}_Z$, of a zone Z is the unique clock valuation of Z satisfying $\forall v \in Z. \forall x \in C. \mathbf{0}_Z(x) \leq v(x)$. Using the canonical DBM representation of Z, $\mathbf{0}_Z$ is easily computed.

Definition 5.1 (Linearly Priced Zone) *A linearly priced zone $\mathcal{Z}$ is defined as tuple $(Z, c_{\mathbf{0}}, q)$, where Z is a zone, $c_{\mathbf{0}} \in \mathbb{N}$ describes the cost of the offset, $\mathbf{0}_Z$, of Z, and $q : C \to \mathbb{Z}$ assigns a cost rate $q(x)$ for any clock x. We write $v \in \mathcal{Z}$ whenever $v \in Z$. We define* $\text{cost}(v, \mathcal{Z}) = c_{\mathbf{0}} + \sum_{x \in C} q(x) \cdot (v(x) - \mathbf{0}_Z(x))$, *if $v \in \mathcal{Z}$, and* $\text{cost}(v, \mathcal{Z}) = \infty$, *otherwise.*

Thus, the cost assignments of a priced zone define a linear plane over the underlying zone and may alternatively be described by a linear expression over the clocks. Figure 5.2 illustrates the priced zone $\mathcal{Z} = (Z, c_{\mathbf{0}}, p)$ over the clocks $\{x, y\}$, where Z is given by the six constraints $2 \leq x \leq 7$, $2 \leq y \leq 6$ and $-2 \leq x - y \leq 3$, the cost of the offset $\mathbf{0}_Z = (2, 2)$ is $c_{\mathbf{0}} = 4$, and the cost-rates are $q(x) = -1$ and $q(y) = 2$. Hence, the cost of the clock valuation $(5.1, 2.3)$ is given by

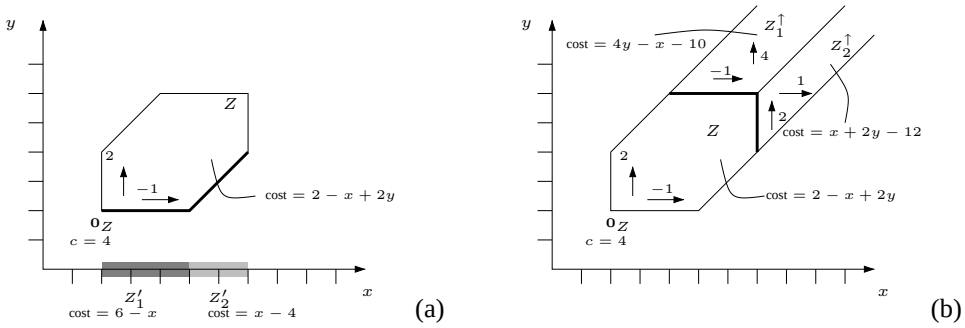


Figure 5.2: A linearly priced zone and its successor sets. (a) The grey shaded areas on the x-axis depicts the successors of a reset of clock y. (b) The delay successors in a location with cost rate 3. The delay successor $({}^{\uparrow}Z)$ is the union of Z, $Z_1^{\uparrow}$ and $Z_2^{\uparrow}$.

$4 + (-1) \cdot (5.1 - 2) + 2 \cdot (2.3 - 2) = 1.5$. In general the costs assigned by $\mathcal{Z}$ may be described by the linear expression $2 - x + 2y$.

Linearly Priced zones can be seen as straightforward extension of priced regions. In analogy with Definition 3.4, we may partition the set of clocks C for a given zone into subsets $r_0, \ldots, r_k$ such that for all $v \in \mathcal{Z}$ the following holds:

1. $x \in r_0 \Leftrightarrow \exists n \in \mathbb{N}.\ v(x) = n$
2. $x, y \in r_i \Leftrightarrow \exists m \in \mathbb{Z}.\ v(x) - v(y) = m$

Obviously, this partition defines an equivalence relation on C. Since $\mathbf{0}_Z \in \mathcal{Z}$ by definition, $v(x) = \mathbf{0}_Z(x)$, if $x \in r_0$. We also have $v(x) - \mathbf{0}_Z(x) = v(y) - \mathbf{0}_Z(y)$, if $x, y \in r_i$. As a consequence, the cost of a valuation $v \in \mathcal{Z}$ can be written as:

$$\text{cost}(v, \mathcal{Z}) = c_{\mathbf{0}} + \sum_{i=1,\ldots,k} (\Sigma_{y \in r_i} q(y))(v(x_i) - \mathbf{0}_Z(x_i)) \tag{5.1}$$

where x_i is some clock in r_i. This equation implies that the cost rate $q(x)$ does not influence the cost of v if $x \in r_0$. It also implies that only the sum of the cost rates of equivalent clocks matters. This fact will be used in the next section, to ensure that operations are independent of the choice of a particular equivalent clock.

The representation of priced regions in Section 3.3 relates to the representation of zones as follows. Given a priced region $R = (h, [r_0, \ldots, r_k], [c_0, \ldots, c_k])$, the subsets $r_0, \ldots, r_k$ constitute a partition as defined prior to Equation (5.1). We can identify $\mathbf{0}_Z$ with h, $c_{\mathbf{0}}$ with c_0, $\text{frac}(v(x_i))$ with $v(x_i) - \mathbf{0}_Z(x_i)$ and finally, $q(x_{k-i})$ with $c_{i+1} - c_i$, for $i = 0, \ldots, k-1$ and $x_{k-i} \in r_{k-i}$. Actually, the machinery developed for priced regions works as well, if we would replace the absolute costs

in the vertices of a region by cost rates. Absolute cost, however, allows to prove termination of the algorithm more easily. For the cost is well founded, it follows that the comparison on priced regions defines a well quasi order.

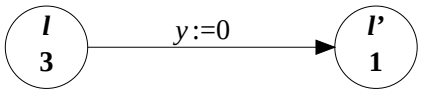


Figure 5.3: A small LPTA

Priced symbolic states are represented in the obvious way by pairs $(l, \mathcal{Z})$, where l is a location and $\mathcal{Z}$ a linearly priced zone. Unfortunately, linearly priced symbolic states are *not* directly closed under the delay and reset operation. To see this, consider the small LPTA in Figure 5.3, with two locations l and l' and a single edge e from l to l' with trivial guard $true$ and reset of clock y. The cost rate of l is 3 and the transition has zero cost. Now, let $\mathcal{Z} = (Z, c_{\mathbf{0}}, r)$ be the priced zone depicted in Figure 5.2(a) and consider the associated priced symbolic state $(l, \mathcal{Z})$.

Assuming that the set of successors of all states $(l, v) \in (l, \mathcal{Z})$ is again expressible as a single priced symbolic state $(l', \mathcal{Z}')$, would obviously require $\mathcal{Z}' = (Z', c'_{\mathbf{0}}, r')$ with $Z' = \{y\}Z$. Following our framework of Section 4.2, the cost-assignment of $\mathcal{Z}'$ should be such that $\text{cost}(v', \mathcal{Z}') = \inf\{\text{cost}(v, \mathcal{Z}) \mid v \in \mathcal{Z} \wedge v[y \mapsto 0] = v'\}$ for all $v' \in \mathcal{Z}'$. Since $q(y) > 0$, it is obvious that these infima are obtained along the lower boundary of Z with respect to y (see Figure 5.2 (a)). In particular, $\text{cost}((2,0), \mathcal{Z}') = 4$, $\text{cost}((4,0), \mathcal{Z}') = 2$, and $\text{cost}((6,0), \mathcal{Z}') = 2$. In general $\text{cost}((x,0), \mathcal{Z}') = \text{cost}((x,2), \mathcal{Z}) = 6 - x$ for $2 \leq x \leq 5$ and $\text{cost}((x,0), \mathcal{Z}') = \text{cost}((x, x-3), \mathcal{Z}) = x - 4$ for $5 \leq x \leq 7$. Obviously, the desired cost-assignment is *not* linear and hence not obtainable from any *single* linearly priced zone. On the other hand, it is also shows that splitting $Z' = \{y\}Z$ into the sub-zones $Z'_1 = Z' \wedge 2 \leq x \leq 5$ and $Z'_2 = Z' \wedge 5 \leq x \leq 7$, allows to represent the successors as the union of *two* priced zones with $q(x) = -1$ in Z'_1 and $q(x) = 1$ in Z'_2).

Similarly, priced symbolic states are *not* directly closed under the delay operation. To see this, consider again the LPTA from Figure 5.3 and the priced zones $\mathcal{Z} = (Z, c_{\mathbf{0}}, r)$ depicted in Figure 5.2(b). Clearly, the set of delay successors must cover the zone $Z^{\uparrow}$. First, we have assign a cost to valuations $(x, y) \in Z$. To do so it is crucial to compare the cost-rate of l (here $P(l) = 3$) with the sum of clock cost-rates of $\mathcal{Z}$ (here $q(x) + q(y) = 1$). Clock valuations (x, y) in Z can obviously be reached by delay from all valuations within Z of the form $(x - \epsilon, y - \epsilon)$, $\epsilon \in \mathbb{R}_{\geq 0}$. The cost of $(x - \epsilon, y - \epsilon)$ is $2 - x + 2y - \epsilon$. If we delay $(x - \epsilon, y - \epsilon)$ with ϵ time units, we will reach (x, y) with cost $2 - x + 2y + 2\epsilon$. Since the cost-rate of l exceeds $q(x) + q(y)$, the minimum cost is obtained when $\epsilon = 0$.

Similarly, the clock valuations (x, y) in $Z^{\uparrow}\backslash Z$ are reached most cheaply from Z by delaying from the *upper* boundary of Z, i.e. from valuations $(x, 6)$ or $(7, y)$ depending on whether $x - y \leq 1$ or $x - y \geq 1$ (see Figure 5.2 (b)). The resulting cost are $4y - x - 10$ and $x + 2y - 12$, respectively. It can be seen that, although the delay successors is not representable by a *single* priced symbolic state, it may be expressed as a *finite union* by splitting the zone $Z^{\uparrow}$ into the three sub-zones Z, $Z_1^{\uparrow} = (Z^{\uparrow}\backslash Z) \wedge (x - y \leq 1)$, and $Z_2^{\uparrow} = (Z^{\uparrow}\backslash Z) \wedge (x - y \geq 1)$ [1]. Let $(l, \mathcal{Z})$ be a symbolic state of an LPTA $A = (\mathit{Loc}, l_0, E, \mathit{Inv}, P)$, we then define for our convenience $Post_{\delta}((l, \mathcal{Z}))$ as set of symbolic delay successors of $(l, \mathcal{Z})$, and $Post_e(l, \mathcal{Z})$ as set of symbolic successors of a transition $e \in E$. The next section presents the operations that are necessary to compute these successors.

5.3 Facets & Operations on Linearly Priced Zones

The key to expressing successor sets of priced symbolic states as finite unions is provided by the notion of *facets* of a zone Z. Formally, whenever $x \sim n$ or $x - y \sim m$ is a constraint of Z, the strengthened zone $Z \wedge (x = n)$ or $Z \wedge (x - y = m)$, respectively, is a facet of Z. Facets derived from lower bounds on individual clocks, $x \geq n$, are classified as *lower facets*, and we denote by $LF(Z)$ the collection of all lower facets of Z. Similarly, the collection of *upper facets*, $UF(Z)$, of a zone Z is derived from upper bounds of Z. We refer to lower as well as upper facets as *individual clock* facets. Facets derived from lower bounds of the forms $x \geq n$, $x - y \geq m$, and $y - x \leq m$ with $n \in \mathbb{N}$ and $m \in \mathbb{Z}$, are classified as lower *relative facets* w.r.t. x. The collection of lower relative facets of Z w.r.t. x is denoted $LF_x(Z)$. The collection of upper relative facets of Z w.r.t. x, $UF_x(Z)$, is derived similarly. Figure 5.4(left) illustrates a zone Z together with its six facets: e.g. Z_1 and Z_6 are the lower facets of Z, and Z_1, and Z_2 the lower relative facets of Z w.r.t. y.

The importance of facets comes from the fact that they allow for decompositions of the delay- and reset-operations on zones as follows:

Lemma 5.2 *Let Z be a zone and x a clock. Then the following holds:*

i) $Z^{\uparrow} = \bigcup_{F \in LF(Z)} F^{\uparrow}$ iii) $\{x\}Z = \bigcup_{F \in LF_x(Z)} \{x\}F$

ii) $Z^{\uparrow} = Z \cup \bigcup_{F \in UF(Z)} F^{\uparrow}$ iv) $\{x\}Z = \bigcup_{F \in UF_x(Z)} \{x\}F$

Proof As all facets are a subset of Z, the $\supseteq$ direction follows in all cases immediately from the definition of delay and reset. To prove $\subseteq$ we introduce the following

[1] $Z_1^{\uparrow}$ is formally not a zone in our sense as we do not allow strict inequalities.

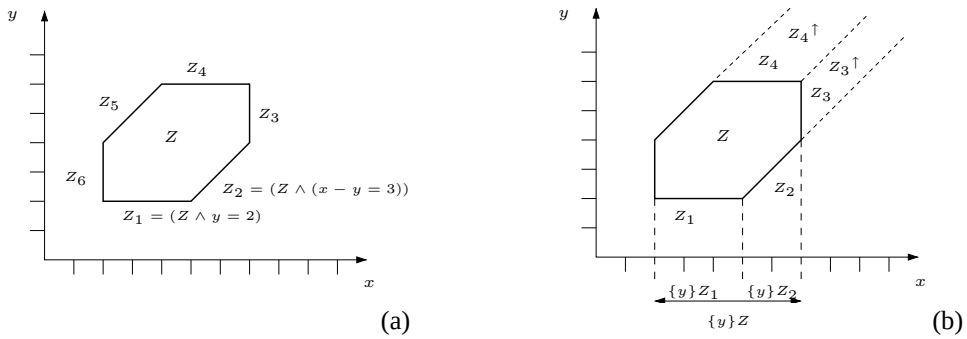


Figure 5.4: A linearly priced zone: Facets and operations.

notation: Given the set of constraints that define a zone Z, we denote with $LB(Z)$ the set of lower bounds $x \geq n$, and with $UB(Z)$ the set of upper bound $x \leq n$, where $n \in \mathbb{N}$. $LB_x(Z)$ denotes, similarly to relative facets, the set of lower bounds $x - y \geq m$, $y - x \leq m$, and $x \geq n$ relative to x, and UB_x the set of the upper bound relative to x. We assume without loss of generality that there are no equality constraints.

(i) Let $v \in Z^{\uparrow}$. We show that there exist a delay predecessor that is an element of a lower facet $Z \wedge (x = n)$.
Let $d_{max} = \min\{v(x) - n | (x \geq n) \in LB(Z)\}$. Since $v \in Z^{\uparrow}$ there exists a valuation $v - d \in Z$, with $d \in \mathbb{R}_{\geq 0}$. Obviously $d \leq d_{max}$, otherwise $v - d$ would violate a constraint of Z. Since $v - d \in Z$, it satisfies all constraints of Z, in particular all upper and relative bounds. Hence, we have that $v - d_{max}$ satisfies all upper and relative bounds, too. By the choice of d_{max} valuation $v - d_{max}$ satisfies also all constraints in $LB(Z)$. Hence $v - d_{max} \in Z$. Additionally, we have for some lower bound $x \geq n$ that $v(x) - d_{max} = n$ holds. Therefore, $v - d_{max} \in Z \wedge (x = n)$ as desired.

(ii) Let $v \in Z^{\uparrow}$. We define $d_{min} = \max\{v(x) - n | (x \leq n) \in UB(Z)\}$. Note, $d_{min} \leq 0$ if $v \in Z$; but in this case the $\subseteq$ direction holds trivially. For $v \in Z^{\uparrow}$ there exists $v - d \in Z$, with $d_{min} \leq d$. It follows that $v - d_{min}$ satisfies all lower and relative bounds as successor of $v - d \in Z$. But it satisfies also the upper bounds by the choice of d_{min}. Furthermore, there exists $x \leq n$ such that $v - d_{min} \in Z \wedge (x = n)$.

(iii) Let $v \in \{x\}Z$. Let $d_{min} = \max(\{n | (x \geq n) \in LB_x(Z)\} \cup \{v(y) + m | (x - y \geq m) \in LB_x(Z)\} \cup \{v(y) - m | (y - x \leq m) \in LB_x(Z)\})$. For $v \in \{x\}Z$ there exist a $v[x \mapsto d] \in Z$, with $d \geq d_{min}$. Clearly, $v[x \mapsto d_{min}]$ satisfies

all upper bounds, but by the choice of d_{min} also all lower bounds. Hence, $v[x \mapsto d_{min}] \in Z$. In addition, $v[x \mapsto d_{min}]$ satisfies either a bound $x = n$ or a relative bound $x - y = m$ or $y - x = m$.

(iv) Similar to (iii), by replacing $\max$ with $\min$, $\geq$ with $\leq$, $\leq$ with $\geq$, and upper by lower bounds. □

Informally, Lemma 5.2 i) expresses that any valuation reachable by delay from Z, is also reachable from one of the lower facets of Z. Part ii) states that this holds also for upper facets. Part iii) (and iv)) express that any valuation in the projection of a zone will be in the projection of the lower (upper) facets of the zone relative to the relevant clock.

As a first step, the delay- and reset-operation may be extended in a straightforward manner to linearly priced (relative) facets:

Definition 5.3 *Let $\mathcal{Z} = (F, c_{\mathbf{0}}, q)$ be a linearly priced zone, where F is a relative facet w.r.t. x.*

1. *Suppose $x = n$ is a constraint of F. Then $\mathrm{reset}(x, \mathcal{Z}) = (F', c_{\mathbf{0}}, q)$ with $F' = \mathrm{reset}(x, F)$.*
2. *Suppose $x - y = m$ is a constraint of F. Then $\mathrm{reset}(x, \mathcal{Z}) = (F', c'_{\mathbf{0}}, q')$, where $F' = \{x\}F$, $c' = c$, and $q'(x) = q(x) + q(x)$ and $q'(y) = q(y)$ for $y \neq x$.*

This definition of $\mathrm{reset}(x, \mathcal{Z})$ is somewhat ambiguous since it depends on which constraint involving x is chosen. But, if there are two constraints $x - y = m$ and $x - z = m'$, then clock y and z are equivalent in the sense of Equation (5.1). Since resetting x does not touch the constraint $y - z = m' - m$, both clocks remain equivalent and it does not matter whether to add $q(x)$ to $q(y)$, or $q(z)$. The cost-function on $\mathcal{Z}$ will be independent of this choice. The relation of this definition with the definition of the reset on priced regions is straightforward. Case 1 in this definition correspond to case 1 in Definition 3.7, and case 2 to case 2 of that definition.

Definition 5.4 *Let $\mathcal{Z} = (F, c_{\mathbf{0}}, q)$ be a linearly priced zone, where F is a lower or upper facet in the sense that $x = n$ is a constraint of F. Let $p \in \mathbb{N}$ be a cost rate. Then $\mathrm{delay}(p, \mathcal{Z}) = (F', c'_{\mathbf{0}}, q')$, where $F' = F^{\uparrow}$, $c' = c$, and $q'(x) = p - \sum_{y \neq x} q(y)$ and $q'(y) = q(y)$ for $y \neq x$.*

If there are two constraints $x = n$ and $y = n'$, it does not matter which clock we choose, as long as the sum of all cost rates of the delay successor is equal to p.

This sum determines the rate of the cost along diagonals in zone $\mathcal{Z}$, and has to be equal to the cost rate of the location. This definition of the delay successor of facets corresponds to case 1 in the Definition 3.6 for regions.

Conjunction of constraints may be lifted from zones to linearly priced zones simply by taking into account the possible change of the offset. Formally, let $\mathcal{Z} = (Z, c_{\mathbf{0}}, q)$ be a linearly priced zone and let $g \in \mathcal{B}(C)$. Then $\mathcal{Z} \wedge g$ is the linearly priced zone $\mathcal{Z}' = (Z', c'_{\mathbf{0}}, q')$ with $Z' = Z \wedge g$, $q' = q$, and $c' = \text{cost}(\mathbf{0}_{Z'}, \mathcal{Z})$. For $\mathcal{Z} = (Z, c_{\mathbf{0}}, q)$ and $n \in \mathbb{N}$ we denote by $\mathcal{Z} \oplus n$ the linearly priced zone $(Z, c_{\mathbf{0}} + n, r)$.

The constructs of Definitions 5.3 and 5.4 essentially provide the $Post$-operations for priced facets. More precisely, it is easy to show that:

$$Post_e(l, \mathcal{Z}_1) = (l', \{y\}(\mathcal{Z}_1 \wedge g) \oplus P(e))$$
$$Post_\delta(l, \mathcal{Z}_2) = (l, \text{delay}(P(l), (\mathcal{Z}_2 \wedge I(l))) \wedge I(l))$$

if $e = (l, g, \{y\}, l')$, $\mathcal{Z}_1$ is a priced relative facet w.r.t. to y and $\mathcal{Z}_2$ is an upper or lower facet. Now, the following lemmas extend this result to linearly priced symbolic states in general:

Theorem 5.5 *Let $\mathcal{A} = (L, l_0, E, I, P)$ be an LPTA. Let $e = (l, g, \{y\}, l') \in E$[2] with $P(e) = p$, $I(l) = J$ and let $\mathcal{Z} = (Z, c_{\mathbf{0}}, q)$ be a linearly priced zone. $Post_e((l, \mathcal{Z}))$ is then equal to:*

$$\{\, (l', \{y\}Q \oplus p) \mid Q \in LF_y(\mathcal{Z} \wedge g) \,\} \quad \text{if } r(y) \geq 0$$
$$\{\, (l', \{y\}Q \oplus p) \mid Q \in UF_y(\mathcal{Z} \wedge g) \,\} \quad \text{if } r(y) \leq 0$$

Theorem 5.6 *Let $\mathcal{A} = (L, l_0, E, I, P)$ be an LPTA. Let $P(l) = p$, $I(l) = J$ and $\mathcal{Z} = (Z, c_{\mathbf{0}}, q)$ be a linearly priced zone. The set of delay successors $Post_\delta((l, \mathcal{Z}))$ is then equal to:*

$$\{(l, \mathcal{Z})\} \cup \{\, (l, \text{delay}(p, Q) \wedge J) \mid Q \in UF(\mathcal{Z} \wedge J)\} \quad \text{if } p \geq \textstyle\sum_{x \in C} r(x)$$
$$\{\, (l, \text{delay}(p, Q) \wedge J) \mid Q \in LF(\mathcal{Z} \wedge J)\} \quad \text{if } p \leq \textstyle\sum_{x \in C} r(x)$$

In the definition of $Post_e$ the successor set is described as a union of either lower or upper relative facets w.r.t. to the clock y being reset, depending on the rate of y (as this will determine whether the minimum is obtained at the lower of upper boundary). For similar reason, in the definition of $Post_\delta$, the successor-set is expressed as a union over either lower or upper (individual clock) facets depending on the rate of the location compared to the sum of clock cost-rates.

[2] In the case of a general reset-set r, the notion of relative facets may be generalized to sets of clocks.

To complete the instantiation of the framework of Section 4.2, it remains to be shown how to compute mincost and $\sqsubseteq$ on priced symbolic states. Let $\mathcal{Z} = (Z, c_{\mathbf{0}}, q)$ and $\mathcal{Z}' = (Z', c'_{\mathbf{0}} q')$ be linearly priced zones and let $(l, \mathcal{Z})$ and $(l', \mathcal{Z}')$ be corresponding priced symbolic states. Then mincost$(l, \mathcal{Z})$ is obtained by minimizing the linear expression $c + \sum_{x \in C}(q(x) \cdot (x - \mathbf{0}_{\mathcal{Z}}(x))$ under the (linear) constraints expressed by Z. Thus, computing mincost reduces to solving a Linear Programming problem.

To define comparison we follow the philosophy that a linearly priced zone is better than another if it is as big and as cheap as the other. Let $\mathcal{Z} = (Z, c_{\mathbf{0}}, q)$ and $\mathcal{Z}' = (Z', c'_{\mathbf{0}}, q')$. We define $\mathcal{Z} \sqsubseteq \mathcal{Z}'$ iff $Z' \subseteq Z$ and cost$(v, \mathcal{Z}) \leq$ cost$(v, \mathcal{Z}')$ for all $v \in Z'$. The last requirement is equal to

$$c'_{\mathbf{0}} - c_{\mathbf{0}} + \sum_{x \in C}(q'(x) - q(x))(v(x) - \mathbf{0}_{\mathcal{Z}}(x)) \geq 0$$

for all $v \in Z'$, which can again be reduced to solving a Linear Programming problem.

Termination of the algorithm of Figure 4.2 can be shown as in Section 3.6 by translating the LPTA to its bounded equivalent. The algorithm considers linearly priced zones $\mathcal{Z}$ with non-negative cost assignments in the sense that cost$(v, \mathcal{Z}) \geq 0$ for all $v \in \mathcal{Z}$. Now, application of Higman's Lemma [Hig52] ensures that $\sqsubseteq$ is a well-quasi ordering on priced symbolic states for bounded LPTA. We thus cannot find an infinite sequence of linearly priced zones that are all incomparable.

5.4 Implementation & Experiments

In this section we give further details on a prototype implementation within the tool UPPAAL [LPY97] of priced zones, formally defined in the previous sections, and reports on experiments on the aircraft landing problem and other examples conducted with the prototype tool. The prototype implements the $Post_e$ (reset), $Post_\delta$ (delay), mincost, and $\sqsubseteq$ operations, using extensions of the DBM algorithms outlined in [Rok93]. To minimize the number of facets considered and reduce the size of the LP problems needed to be solved, we make heavy use of the canonical representation of zones in terms of a *minimal* set of constraints given in [LLPY97]. For dealing with LP problems, our prototype currently uses a freely available implementation of the simplex algorithm.[3]

Many of the techniques for pruning and guiding the state-space search described in Chapter 4 are directly applicable to the algorithm in Figure 4.2, i.e. pruning the state-space according to the variable COST, computing a lower bound

[3] lp_solve 3.1a by Michael Berkelaar, `ftp://ftp.es.ele.tue.nl/pub/lp_solve`.

Table 5.5: Results for seven instances of the aircraft landing problem. Results were obtained on a Pentium II 333MHz.

run-ways	problem instance	1	2	3	4	5	6	7
	number of planes	10	15	20	20	20	30	44
	number of types	2	2	2	2	2	4	2
1	optimal value	700	1480	820	2520	3100	24442	1550
	explored states	481	2149	920	5693	15069	122	662
	cputime (secs)	4.19	25.30	11.05	87.67	220.22	0.60	4.27
2	optimal value	90	210	60	640	650	554	0
	explored states	1218	1797	669	28821	47993	9035	92
	cputime (secs)	17.87	39.92	11.02	755.84	1085.08	123.72	1.06
3	optimal value	0	0	0	130	170	0	
	explored states	24	46	84	207715	189602	62	N/A
	cputime (secs)	0.36	0.70	1.71	14786.19	12461.47	0.68	
4	optimal value				0	0		
	explored states	N/A	N/A	N/A	65	64	N/A	N/A
	cputime (secs)				1.97	1.53		

on the remaining cost, exploring states in minimum cost order, and using heuristics to quickly guide the search to a goal state.

Example [Aircraft Landing Problem (continued)] Recall the aircraft landing problem partially described in the introduction. An LPTA model of the costs associated with landing a single aircraft is shown in Figure 5.1(b). When landing several planes the schedule has to take into account the separation times between planes to avoid the turbulence of one plane affecting another. The separation times depend on the types of the planes that are involved. Large aircrafts for example generate more turbulence than small ones, and successive planes should consequently keep a bigger distance, if it is preceded by large aircraft.

The LPTA in Figure 5.1(c) models the separation times between two types of planes. The automaton has two clocks `c1` and `c2` to measure the time since the last plane of type 1 or 2, respectively has landed. The guard `c1>=wait21` then ensures that the separation time between a plane of type 2 and a plane of type 1 is bigger than constant `wait21`.

Table 5.5 presents the results of an experiment that applies the prototype to seven instances of the aircraft landing problem taken from [BKA00][4]. For each instance, which varies in the number of planes and plane types, we compute the cost of the optimal schedule. In case the cost is non-zero we increase the number of runways until a schedule of cost 0 is found. This is always possible as the cost of landing on target time is 0 and the number of runways can be increased until all planes arrive at target time. In all instances, the state-space is explored in minimal-

[4] These and other benchmarks are available at `ftp://mscmga.ms.ic.ac.uk/pub/`.

cost order, i.e. we select from the waiting list the priced zone $(l, \mathcal{Z})$ with lowest mincost$(l, \mathcal{Z})$. Equal values are distinguished by selecting first the zone which results from the largest number of transitions, and secondly by selecting the zone which involves the plane with the smallest target time.

As can be seen from the table, our current prototype implementation is able to deal with all the tested instances. Beasley et al. [BKA00] solve all problem instances with a linear programming based tree search algorithm, in cases that the initial solution – obtained with a heuristic – is not zero. In 7 of the 15 benchmarks (with optimal solution greater than zero) the time-performance of our method is better than theirs. These are the instances 4 to 7 with less than 3 runways. This result also holds if we take into account that our computer is about 50% faster (according to the Dongarra Linpack benchmarks [Don01]).

It should be noted, however, that our solution-times are quite incomparable to those of Beasleys. For some instances our approach is up to 25 times slower, while for others it is up to 50 times faster than the approach in [BKA00]. One reason for these results is that Beasley et al. solve less but larger LP-problems. Our models of the aircraft landing problems have usually less than 10 clocks. The priced zones therefore cannot have more than 110 constraints. The LP-based tree search in contrast results in problems with up to 600 variables and 1200 constraints. □

Example [Extended Bridge Problem] To extend the problem from Section 4.5.2 we introduce a cost associated with staying on the unsafe side of the bridge. The problem is now to find a way for the four persons to cross the bridge which results in the lowest possible cost.

Table 5.6: Schedules and minimum costs for cost extended versions of the bridge problem.

Cost-rates				**Schedule**					**Cost**	**Time**	**States**
A_5	B_{10}	C_{20}	D_{25}						-		
Min Time				BA	A	CD	B	BA	-	60	1715
Min Time				BA	A	CD	B	BA	-	60	(1536)
1	1	1	1	BA	A	CA	A	AD	55	65	301
9	2	3	10	AD	A	BA	A	CA	195	65	144
1	2	3	4	BA	A	CD	B	BA	140	60	208
1	2	3	10	CD	C	BC	B	BA	165	85	262

Table 5.6 depicts the minimum costs for five instances of the extended bridge problem. For each instance we give the four costs assigned to the persons for residing on the initial side, the schedule (in which step 2 and 4 always represent a single person crossing the bridge back to the initial side), the minimum cost, the

time of the generated schedule with minimum cost, and the number of explored states. All results have been obtained by searching the state-space in minimal-cost order.

The first result in Table 5.6 were measured with a model for finding a time-optimal solution to the problem. The results on the first line shows that the time-optimal schedule requires 60 minutes and that 1715 (symbolic) states are explored to find the solution with the cost-extended version of UPPAAL based on linearly priced zones. The second line shows (within parenthesis) the number of symbolic states need to solve the same problem with the version presented in Chapter 4 for UPTAs. Thus, in comparison the general cost version increments the number of explored states with less than 15% in this example. The prototype for UPTAs is, however, with 0.2 seconds cpu time on a Pentium III 500MHz much faster than the prototype for general LPTAs, that needs 8 seconds to solve the minimal time problem. The four last lines of the table show the number of explored states when optimizing costs instead of time. From these results we observe that considering general costs seems to significantly reduce the number of explored symbolic states. □

Example [Others] In the optimal broadcast problem, UPPAAL is applied to find schemes for broadcasting messages in a network consisting of several routers connected with two communication channels. The cost and time of using the two channels differ and the problem is to find a time or cost-optimal schedule for broadcasting a set of messages to all subscribed routers. So far, we have been able to solve this problem (with varying communication costs) for up to six routers.

In the testing example, the problem is to find a minimal set of test sequences to fully cover different aspects of the sender component of the audio protocol in [BGK+96]. This can be done by annotating the model with edges and testing variables which are set when an aspect of the specification has been covered. The cost extended version of UPPAAL can then be applied to find the cheapest possible path trough the specification which sets all the testing variables. We have been able to apply this technique in UPPAAL to generate optimal testing sequences for covering e.g. all location, all synchronization actions, or all edges of the protocol specification. □

5.5 Conclusion

In this chapter we have considered the minimum-cost reachability problem for LPTAs. The notions of linearly priced zones, and facets of a zone are central con-

tributions of the chapter underlying our extension of the tool UPPAAL. Our initial experimental investigations are quite encouraging.

Compared with the existing special-purpose, time-optimizing version of UPPAAL [BFH+01a], the presented general cost-minimizing implementation does only marginally down-grade performance in the number of explored states. In particular, the theoretical possibility of uncontrolled splitting of zones does not occur in practice. In addition, the consideration of non-uniform cost seems to significantly reduce the number of symbolic states explored.

The single, most important question, which calls for future research, is how to exploit the simple structure of the LP-problems considered. In our approach we for example encounter, for example, only constraints of the form $x - y \sim m$ and $x \sim n$, with $\sim \in \{\leq, =, \geq\}$. We may benefit significantly from replacing the currently used LP package with some package that is tailored towards small-size problems of this kind.

6

Guiding Polyhedral Reachability Analysis of Hybrid Systems

6.1 Introduction

Hybrid systems are discrete event systems that interact with a continuous environment, typically a discrete controller of an analog system. The hybrid automata model provides a framework to specify and analyze hybrid systems. Hybrid automata distinguish, like timed automata, between discrete transitions and continuous transitions. But unlike timed automata, which allow only clocks with rate one, hybrid automata may describe the evolution of the continuous variables by any kind of differential equation.

To compute all reachable states of a hybrid system one typically searches exhaustively for new symbolic states until a fixpoint is reached. A symbolic state is a pair of a discrete state (or control location) and a set of continuous states. But unlike timed automata, hybrid automata provide in general no (finite) partition of the state-space which is closed under taking transitions – either discrete or continuous ones. There are several approaches that use over-approximations to overcome this problem. Examples are approximations with orthogonal polyhedra [DM98], projections to lower dimensional polyhedra [GM99], ellipsoids [KV00], or bounded polyhedra [CK99, Var98, Feh98].

Another approach to this problem is to restrict the continuous behavior of the hybrid system such that it becomes suitable to use the exact sets of reachable states. The model checker HyTech offers a semi-decidable reachability algorithm for the class of linear hybrid systems [HHWT95]. If the algorithm stops

This chapter covers the publications:

[Feh98] A. Fehnker. *Automotive Control Revisited – Linear Inequalities as Approximation of Reachable Sets.* HSCC'98, 1998.

[Feh00b] A. Fehnker. *Heuristic Reachability Analysis of Hybrid Systems,* Manuscript, *2000.*

the exploration, the computed set of reachable states is exact. In addition there are restricted classes of hybrid automata for which the reachability problem is decidable [HKPV95, LPY99], i.e. it is possible to construct a finite partition of the state-space which is closed under taking transitions. The classes of decidable and semi-decidable hybrid systems can be useful, even if a hybrid system is not from these classes. Rather than approximating the reachable sets, one can analyze an approximate model of the hybrid system [HH95].

This chapter presents an approach which was put forward in [Feh98], and which is similar to the approach proposed independently in [Var98]. This approach uses optimal control theory to approximate the reachable sets of hybrid systems with uncertain input. The over-approximation was then used in [Feh98] to analyze an automotive control problem. As an extension to this earlier work we present and evaluate in this chapter a model checking algorithm similar to the timed automata algorithm in Chapter 3 that allows the use of heuristic search orders.

This chapter deals (unlike the previous chapters) with the full exploration of the state-space. Also in this case the search order can matter. Exploring first symbolic states that include others can reduce the overall number of states encountered during the search. We will see that even more subtle interactions between the search order and the performance take place. We show for example that some search orders may lead to starvation of symbolic states on the wait list, a phenomenon that does not occur in the case of timed automata. If the algorithm searches, for example, depth-first – generated states are explored in a last-in-first-out fashion – it may happen that a state has been generated but will never be explored. Starvation can also occur with other heuristic search orders and causes non-termination of the algorithm.

In this chapter we consider two case studies. The first one is an automotive suspension system that was introduced by T. Stauner et al. in [SMF97], and which was already dealt with in [Feh98]. The electronic height control (EHC) has to keep the distance between chassis and the wheel of a car within bounds. The height of the chassis is controlled by pneumatic suspension. The level can be increased by pumping air into the system and it can be lowered by opening an escape valve. The height is measured by a low-pass filter that filters disturbances caused for example by holes in the road. As a consequence it takes also some time until changes in height are properly detected.

The second case study arose from an experimental setup used in Computing Science courses at the University of Nijmegen [Kra00]. This setup is made up of a train, a train gate and a car, all built from LEGO and controlled by LEGO MINDSTORM RCX bricks. The car is equipped with two light sensors which allow it to follow a black line on a white floor. Though it is a rather small example it exhibits interesting non-trivial hybrid behavior.

The next section briefly introduces hybrid automata. In Section 6.3 we show how to compute the reachable sets. Section 6.4 presents an algorithm for reachability analysis, and heuristics that select the biggest zones first. In this section we give an example of an hybrid automaton that may lead to starvation, and show how to prevent this. The two case studies are described in more detail in Section 6.5. Section 6.6 presents the experimental results which show that heuristics can improve the performance, and that avoiding starvation may contribute, too. Section 6.7 concludes this chapter.

6.2 Clocked Hybrid Automata

This chapter deals only with small examples that illustrate the use of polyhedra, and the benefits of heuristics. Therefore, we consider only a restricted class of hybrid automata, and omit the discussion of important issues such as compositionality, deadlock, and zeno-ness.

Let $v = (v_1, \ldots, v_n)^T$ be a vector of n continuous variables that range over $\mathbb{R}$. We write V for the set of variables $\{v_1, \ldots, v_n\}$. The set of all linear inequalities of the form $\mathbf{C}\, v \leq b$ with $\mathbf{C} \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ is denoted by $\mathcal{I}(V)$. We will identify a polyhedron $\mathbf{P}$ with the inequalities $\phi \in \mathcal{I}(V)$ which define the polyhedron. Note that different inequalities may define the same polyhedron. We call $\mathbf{v} \in \mathbb{R}^n$ an element of $\mathbf{P}$, if $\mathbf{v} \models \phi$. Let $\mathcal{B}(V)$ be the subset of $\mathcal{I}(V)$ that define bounded polyhedra. Bounded polyhedra are also known as polytopes.

Definition 6.1 *We consider hybrid automata that can be defined by*

- *a set Loc of control locations,*
- *a set of continuous variables V. A pair $(l, \mathbf{v})$ with $l \in$ Loc and $\mathbf{v} \in \mathbb{R}^n$ will be called a state. We divide the set of continuous variables V in sets of* locally controllable *variables X and* input variables *U,*
- *a set Act of labels,*
- *an initial location l_0 together with an initial constraint $\phi_0 \in \mathcal{B}(V)$ on the continuous variables,*
- *an invariant Inv : Loc $\rightarrow \mathcal{I}(V)$,*
- *a set E of discrete transitions of the form $e = (l, \phi, a, \rho, l')$ with $l, l' \in$ Loc, guard $\phi \in \mathcal{I}(V)$, $a \in$ Act, a reset set $\rho \subseteq V$,*
- *and a set of* trajectories *$\mathcal{T}$ over V. A trajectory τ is a mapping from $\mathbb{R}_{\geq 0}$ to states.*

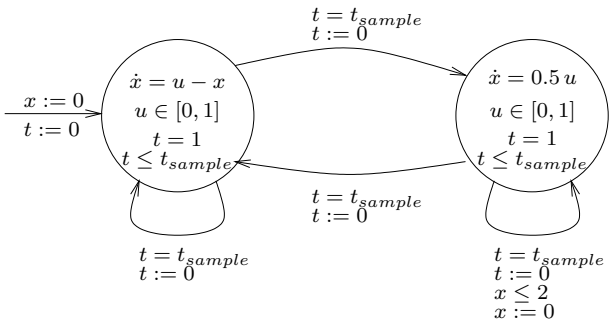


Figure 6.1: A simple hybrid automaton, with uncertain input and clocked transitions.

A general introduction to hybrid automata is to be found in for example [Hen96] and [LSV01].

Furthermore, we assume in the remainder of this chapter that the following holds:

- The continuous behavior in each location of the hybrid automata is specified by a linear, time invariant differential equations of the following form

$$\dot{x}(t) \quad = \quad \mathbf{A}\, x(t) + \mathbf{B}\, u(t) \tag{6.1}$$

  In Control Theory x is known as state of the system, and u called the input. We assume that the input $u : \mathbb{R}_{\geq 0} \rightarrow \mathbf{U}$ is a measurable function, with a bounded range $\mathbf{U} \in \mathcal{B}(U)$.

- The hybrid system has only a finite set of locations, a finite set of discrete transitions and a finite set of continuous variables.

- The systems is *clocked* with sampling time t_{sample}. This means that discrete transitions may only occur every t_{sample} time units. We introduce a clock variable t, and assume that each transition is guarded by $t = t_{sample}$ and resets t.

The last requirement restricts the class to systems in which the discrete controller samples the input regularly. The environment will not change its dynamics by itself, but only as a consequence of controller output. The main reason for this restriction is that it allows to easily implement a prototype model checker that is applicable to the considered case studies. We show also how to approximate reachable sets over intervals of time, which is necessary if we want to deal with a larger class of hybrid systems.

The next section discusses some basic properties of linear equalities. We will derive an approximation method that uses ideas from optimal control theory. Section 6.4 then presents a forward reachability algorithm.

6.3 Approximation of Reachable Sets

Reachability analysis of a hybrid system requires determination of the successors of a state due to the continuous evolution of the system and due to discrete transitions of the controller. Many practical systems use conjunctions of linear inequalities to define guards on transitions. Polyhedra, which are defined by linear inequalities, have some useful properties. A nonempty intersection of two polyhedra is a polyhedron, and a nonempty intersection of a polytope with a polyhedron yields a polytope. Therefore, it is a natural choice to use polyhedra also to approximate sets of successors. We refer to a bounded polyhedron that approximates a reachable set as a *zone*. In contrast to timed automata we require that zones are bounded.

To compute a zone that over-approximates the states that are reachable at a certain time point, starting from a compact (bounded and closed) and convex set of initial states we need some control theory. The unique solution of the differential equation (6.1) with initial state $x(0) \in \mathbf{X}_0$ is given by [1]

$$x(t) = e^{\mathbf{A}\,t} x(0) + \int_0^t e^{\mathbf{A}\,(t-\sigma)} \mathbf{B}\, u(\sigma) d\sigma \tag{6.2}$$

We abbreviate the right-hand side by $\varphi(x_0, t, q)$.

In Control Theory one often wants to find a time optimal control for the system (6.1), assuming $u : \mathbb{R}_{\geq 0} \rightarrow \mathbf{U}$ and $x(0) \in \mathbf{X}_0$, with $\mathbf{U} \subset \mathbb{R}^m$ and $\mathbf{X}_0 \subset \mathbb{R}^n$ compact and convex sets. Let $Reach(\mathbf{X}_0, t_f, \mathbf{U})$ denote the set of states that can be reached from the initial set $\mathbf{X}_0$ at time t_f with inputs from $\mathbf{U}$. Denote the boundary of a set S by $\delta(S)$.

The set of reachable states $Reach(\mathbf{X}_0, t_f, \mathbf{U})$ form under these assumptions a convex and compact set. Let $\mathbf{X}_f$ be $Reach(\mathbf{X}_0, t_f, \mathbf{U})$. Assume that we have $x_f \in \delta(\mathbf{X}_f)$, then there exists a supporting hyper-plane (tangent plane) that contains x_f. Let c_f be the normal on this hyper-plane, then $c_f^T\, x \leq c_f^T\, x_f$ holds for all reachable states $x \in \mathbf{X}_f$. Figure 6.2 sketches this situation. The reachable set, however, and thus x_f will usually be unknown beforehand. But fortunately, it is possible to find for a given c_f and t_f, an input $\bar{u}$ and an initial state x_0 such that $c_f^T\, x \leq c_f^T\, \varphi(x_0, t_f, \bar{u})$ for all $x \in \mathbf{X}_f$ and $u : \mathbb{R}_{\geq 0} \rightarrow \mathbf{U}$. Similarly, one can

[1] Note that $e^{\mathbf{A}t}$ is a symbolic notation for the fundamental solution of $\dot{x} = \mathbf{A}\, x$

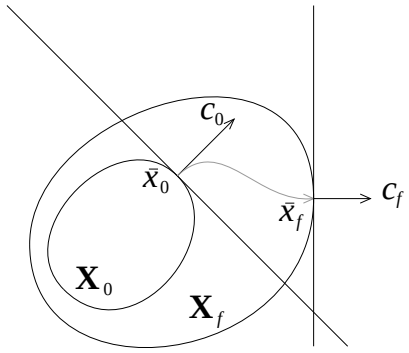


Figure 6.2: Given an initial set $\mathbf{X}_0$, and a normal c_0 one can determine an initial state $\bar{x}_0$ on the boundary and an input u that drives the system from $\bar{x}_0$ to state $\bar{x}_f$ on the boundary of $\mathbf{X}_f$.

determine for a given initial state $x_0 \in \mathbf{X}_0$ with normal c_0, an input $\bar{u}$ and a normal c_f such that the aforementioned inequality holds.

Lemma 6.2 *Suppose $\mathbf{X}_0 \subseteq \mathbb{R}^n$ and $\mathbf{U} \subseteq \mathbb{R}^m$ are convex and compact sets. Let $t_f \in \mathbb{R}_{\geq 0}$ and $c_0 \in \mathbb{R}^n$. Let $c_f := e^{-\mathbf{A}^T t_f} c_0$. Then there exists a $\bar{x}_0 \in \delta(\mathbf{X}_0)$ and a mapping $\bar{u} : \mathbb{R}_{\geq 0} \to \delta(\mathbf{U})$ such that*

$$c_0^T \bar{x}_0 = \max_{x_0 \in \mathbf{X}_0} c_0^T x_0 \tag{6.3}$$

$$c_0^T e^{-\mathbf{A}\,t} \mathbf{B} \bar{u}(t) = \max_{u \in \mathbf{U}} c_0^T e^{-\mathbf{A}\,t} \mathbf{B} u, \quad \forall t \in [0, t_f] \tag{6.4}$$

$$c_f^T \bar{x}_f = \max_{x_f \in \mathbf{X}_f} c_f^T x_f \tag{6.5}$$

with $\bar{x}_f = \varphi(\bar{x}_0, t_f, \bar{u})$ and $\mathbf{X}_f = Reach(\mathbf{X}_0, t_f, \mathbf{U})$.

Equations (6.3) and (6.4) can be established using the fact that there always exists a maximum of a linear function on a compact and convex set (see [LM67]). It should be noted that $\bar{x}_0$ and $\bar{u}$ are not necessarily unique. Using (6.3), (6.4) and (6.2) shows straightforward that $c_f^T(\varphi(\bar{x}_0, t_f, \bar{u}) - \varphi(x_0, t_f, u)) \geq 0$ holds for arbitrary x_0 and u, therefore (6.5) is proven. The relations between x_f, u and x_0 given by this lemma, are used to prove the bang-bang principle (or theorem of Lee-Markus [LM67]). This principle states that it is always possible to reach an extreme state with an extreme control.

We can use these relations to obtain an over-approximation of the reachable sets at time $t_f \in \mathbb{R}_{\geq 0}$. Suppose that we have an initial set $\mathbf{X}_0$, a subset of bounded

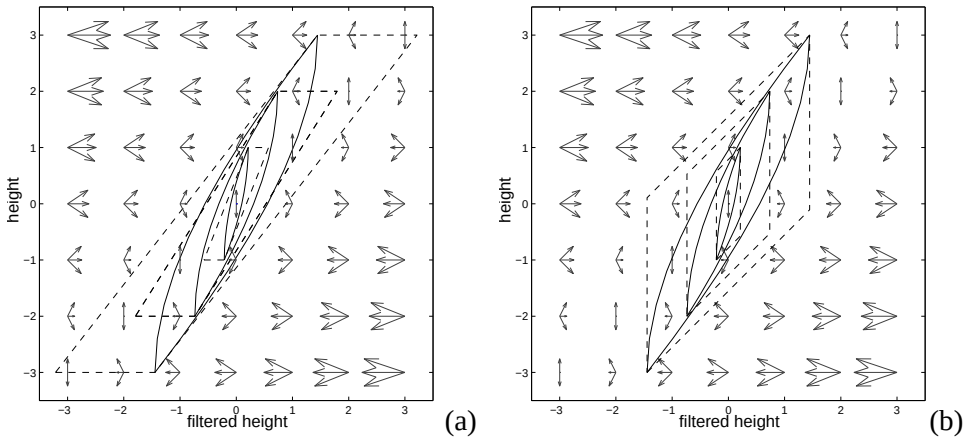


Figure 6.3: The background shows a superposition of three vector fields. The horizontal vectors depict the vector field without disturbance. The two others show the vector field, as result of a maximal disturbance in the vertical directions. The solid lines are the boundaries of the reachable sets, starting from point $(0, 0)$ after 1, 2 and 3 seconds (from in- to outside). The dotted lines are the approximations of these sets with polyhedra. Figure (a) and (b) use different ways to approximate these sets, see text for an explanation. This example is taken from the automotive height control example in Section 6.5.

polyhedron $\mathbf{C}\,x \leq b$. Let c_0^T be a row-vector of matrix $\mathbf{C}$, b_0 the corresponding element of b, and $c_f := e^{-\mathbf{A}^T t_f} c_0^T$. Let x_0 be a point on the boundary of $\mathbf{X}_0$ that maximizes $c_0^T\,x$. According to (6.5) there is an input $\bar{u}$ such that $c_f^T\,x \leq b_f$ for all $x \in \mathbf{X}_f$, with $b_f := c_f^T\,\varphi(\bar{x}_0, t_f, \bar{u})$. In this way one can find a matrix $\mathbf{C}'$ and vector b' such that $Reach(\mathbf{X}_0, t_f, \mathbf{U})$ satisfies $\mathbf{C}'x \leq b'$. This zone can then serve as initial set for the next iteration step. Figure 6.3(a) gives an example of this approximation procedure.

It has still to be shown that the polyhedron $\mathbf{C}'x \leq b'$ is indeed bounded. We will show that $\mathbf{C}'x \leq b'$ infers for any normal c_f a bound on $c_f\,x$, i.e. the polyhedron is bounded in any direction. Let $c_0 := e^{\mathbf{A}^T t_f} c_f$. Since the polyhedron $\mathbf{C}\,x \leq b$ is assumed to be bounded it is possible to determine a vertex p that is optimal with respect to $c_0\,x$. We know – from the termination criterion of the simplex method for example– that it is possible to write c_0 as linear combination $a_1c_1 + \ldots + a_nc_n$ where c_i are proper row-vectors of C and $a_i \in \mathbb{R}_{\geq 0}$. Let c_i' be the corresponding row-vectors of C' and b_i' the bound on $c_i'\,x$. We then have that $c_f = a_1c_1' + \ldots + a_nc_n'$ and thus that if x satisfies the inequality $C'\,x \leq b$ it then satisfies also $c_f\,x \leq a_1b_1 + \ldots + a_nb_n$.

An alternative approach to approximate a set of reachable states chooses first a matrix $\mathbf{C}$. For each row-vector c_f of $\mathbf{C}$ lemma 6.2 allows us to determine c_0, x_0,

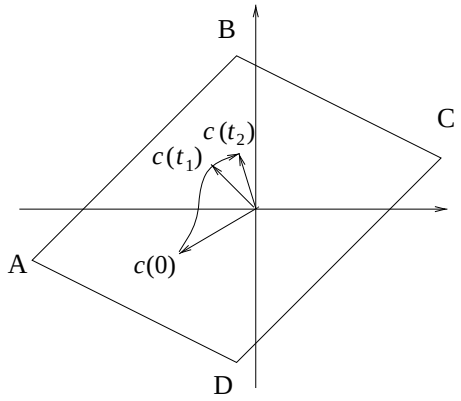


Figure 6.4: The optimum of $c(t)\,x$ may jump from one vertex to another. But not infinitely often in a finite interval.

$\bar{u}$ and consequently x_f, such that $c_f^T\,x \leq b_f$ for all $x \in \mathbf{X}_f$, with $b_f := c_f^T\,\bar{x}_f$. If we repeat this for all row-vectors of $\mathbf{C}$, we obtain linear inequalities $\mathbf{C}\,x \leq b$ that include $Reach(\mathbf{X}_0, t_f, \mathbf{U})$. This alternative approach is depicted in Figure 6.3(b). All approximations use the same matrix $\mathbf{C}$ to define the polyhedra.

This approach has several advantages. Since all zones use the same $\mathbf{C}$, the inclusion check between two zones $\mathbf{C}\,x \leq b_1$ and $\mathbf{C}\,x \leq b_2$ can be reduced to checking inequality of b_1 and b_2. Similarly, checking whether a guard $c\,x \leq b$ is satisfied can be simplified if c is a row-vector of $\mathbf{C}$. Not to mention that we only have to store the bounds b to define a zone, rather than the full inequalities.

The first approach allows to re-use the tangent points as initial values for the next iteration. These are points on the boundary of the exact reachable set, even if the approximation is applied iteratively. The alternative approach cannot guarantee this, since we are not free to choose an initial point once we have chosen c_f. A major disadvantage of the first approach however is, that whenever we take an intersection of a zone with a guard or invariant, this might add new linear equalities to the set of inequalities. In this chapter we use the first approach to analyze the LEGO car, and the second one to analyze the automotive suspension problem.

Often, we are not only interested in the reachable set on certain points in time, but also in constraints on the reachable states in an interval of time. Lemma 6.2 uses the fact that the optimum of a linear function on a compact set will be attained on the boundary. If we take the optimum over a bounded polyhedron, the optimum will be attained in a vertex of the polyhedron. To be able to approximate the reachable set in a time interval we need slightly more.

Lemma 6.3 *Let $\mathbf{P} \in \mathcal{B}(V)$ be a polytope and suppose $c : \mathbb{R}_{\geq 0} \to \mathbb{R}^n$ analytic.*

Then there exist $t_{max} > 0$ and a vertex p of $\mathbf{P}$ such that

$$c(t)^T p = \max_{x \in \mathbf{P}} c(t)^T x \quad \forall t \in [0, t_{max}] \tag{6.6}$$

holds.

This lemma says that there exists a vertex which is optimal at $t = 0$ and stays optimal for at least t_{max} time units. The optimum does not have to be unique, and if two or more vertices are optimal then all points that are a convex combination of these points are also optimal. The proof uses the fact that all functions $c(t)^T p_i$ are analytic. Among these there has to be a function that is greater than or equal to the others on an interval. If this was not the case one could construct a non-constant analytic function which takes the value zero infinitely often on a finite interval.

Figure 6.4 illustrates lemma 6.3 for two dimensions. The maximum (6.6) is attained from 0 up to t_1 in vertex A. At time t_1 the maximum is not unique; it is attained in every point on edge $\overline{AB}$. Without the assumption $c(t)_i$ analytic, one can easily construct a function $c(t)$ (e.g. using $\sin(1/t)$), such that the maximum is not constantly attained in one vertex, for any interval $[0, t_{max}]$.

To get a lemma similar to 6.2, we restrict $\mathbf{X}_0$ and $\mathbf{U}$. In the remainder of this section we assume that both $\mathbf{X}_0$ and $\mathbf{U}$ are bounded polyhedra. We denote with $Reach(\mathbf{X}_0, [0, t_{max}], \mathbf{U})$ the set of states that can be reached from $\mathbf{X}_0$, with input in $\mathbf{U}$, and within time t_{max}.

Lemma 6.4 *Suppose $\mathbf{X}_0 \subseteq \mathbb{R}^n$ and $\mathbf{U} \subseteq \mathbb{R}^m$ are bounded polyhedra. Let c_{max} be a vector in $\mathbb{R}^n$. Then there exits an $\bar{x}_0 \in \delta(\mathbf{X}_0)$, and a $t_{max} > 0$ and a constant $\bar{u} \in \{u | u : [0, t_{max}] \to \delta(\mathbf{U})\}$ with*

$$c_{max}^T e^{\mathbf{A}\,t}\,\bar{x}_0 = \max_{x_0 \in \mathbf{X}_0} c_{max}^T e^{\mathbf{A}\,t} x_0, \quad \forall t \in [0, t_{max}] \tag{6.7}$$

$$c_{max}^T e^{\mathbf{A}\,(t_{max}-t)} \mathbf{B} \bar{u}(t) = \max_{u \in \mathbf{U}} c_{max}^T e^{\mathbf{A}\,(t_{max}-t)} \mathbf{B} u, \quad \forall t \in [0, t_{max}] \tag{6.8}$$

$$c_{max}^T \bar{x}(t) = \max_{x(t) \in \mathbf{X}(t)} c_{max}^T x(t), \quad \forall t \in [0, t_{max}] \tag{6.9}$$

with $\bar{x}(t) = \varphi(\bar{x}_0, t, \bar{u})$ and $\mathbf{X}(t) = Reach(\mathbf{X}_0, t, \mathbf{U})$.

The proof uses the existence of an interval $[0, t_{max1}]$ on which the maximum of $c_{max}^T e^{\mathbf{A}\,t} x_0$ is attained in one vertex $\bar{x}_0$ of $\mathbf{X}_0$ (see lemma 6.3). The same holds for $\bar{u}$; there exists an interval $[0, t_{max2}]$ on which $\bar{u}$ is constant. Take t_{max} as minimum of t_{max1} and t_{max2}. Similar to the proof of lemma 6.2 we use equation (6.7), (6.8) and (6.2) to show that $c_{max}^T(\varphi(\bar{x}_0, t, \bar{u}) - \varphi(x_0, t, u)) \geq 0$ holds for arbitrary x_0,q and all $t \in [0, t_{max}]$, and therefore (6.9) is proven. For $\bar{u}$ is constant on $[0, t_{max}]$ we are able to simplify the integrals which arise from (6.2)

Suppose $\mathbf{X}_0$ is a bounded polyhedron $\mathbf{C}\,x \leq b_0$. Lemma 6.4 allows us to find for a given normal c_f a bound $b(t)$ such that the set $Reach(\mathbf{X}_0, t, \mathbf{U})$ is contained in $c_f^T\, x \leq b(t)$, for all t in interval $[0, t_{max}]$. The upper bounds t_{max1} and t_{max2} depends mostly on $\mathbf{C}$, $\mathbf{A}$, $\mathbf{B}$ and set $\mathbf{U}$. Suppose that the maximum (6.7) is attained in vertex p_1 for $t \in [0, t_{max}]$. Let p_2 be a vertex such that $c_{max}^T e^{\mathbf{A}\,t_{max}}\, p_1 = c_{max}^T e^{\mathbf{A}\,t_{max}}\, p_2$. We then know that edge $\overline{p_1 p_2}$ is perpendicular to $c_{max}^T e^{\mathbf{A}\,t_{max}}$. The orientation of the edges however depends only on matrix $\mathbf{C}$. This allows us to approximate the reachable set even if $t_{sample} > t_{max}$. In this case the approximation technique is applied iteratively to the result of the preceding approximation.

In some cases one needs a single bound on all states which are reachable within interval $[0, t_{max}]$. For this purpose we take the maximum of $b(t)$ over the interval $[0, t_{max}]$. This approximation gets of course worse with a longer interval $[0, t_{max}]$. It should be noted that $\{(x; t) | x \in Reach(\mathbf{X}_0, t, \mathbf{U})\}$ is generally not convex, and hence it is difficult to handle transitions that do not take place at a specified time. The intersection of this set with a guard may yield a non-convex set. Therefore, we restrict the model to clocked systems.

Recall the hybrid automaton in Figure 6.1. The initial set in location loc_1 satisfies $x = 0$. Suppose we want to find a constraint $x \leq b(t)$ on the reachable set. We thus have $c_{max} = 1$. The maximum of $e^{-t}\, x_0$ (equation (6.7)) is attained in vertex $\bar{x}_0 = 0$. With (6.8) we find that $\bar{u} \equiv 1$ on $[0, t_{max}]$, for all $t_{max} > 0$. We then get that all states that can be reached at time t from $x = 0$ with input u from $[0, 1]$ satisfy $x \leq 1 - e^{-t}$. Analogously, we can find the lower bound $0 \leq x$ on all reachable states.

6.4 Reachability Analysis with Heuristics

In order to perform a reachability analysis we have to define a successor relation between symbolic states. We define a symbolic state as a pair $(l, \mathbf{Z})$ of a location $l \in \mathit{Loc}$ and a *zone* $\mathbf{Z} \in \mathcal{B}(V)$. Let $(l, \mathbf{Z})$ be a symbolic state of hybrid automaton A with locations *Loc*, variables V and set of discrete transitions E. Let $approx(l, \mathbf{Z}, t_{sample})$ be the approximation of $Reach(l, \mathbf{Z}, t_{sample})$, using one of the two approaches presented in the previous section. We call state $(l', \mathbf{Z}')$ a successor of $(l, \mathbf{Z})$, if there exist a zone $\mathbf{Z}'' = approx(l, \mathbf{Z}, t_{sample})$ and an action a such that $((l, \mathbf{Z}''), a, (l', \mathbf{Z}')) \in E$. Note that we calculate the successor of a zone as result of a delay and a discrete transition.

Different search orders are realized by means of a *heuristic value* h, which is associated with each symbolic state. We assume that h ranges over the set $\mathbb{R}$. The search order is then defined by a *heuristic function* $H : \mathit{Loc} \times \mathcal{B}(V) \times \mathbb{R} \to \mathbb{R}$, that

```
PASSED := {}
WAITING := [(l_0, Z_0, h_0)]
while WAITING ≠ [] do
   select (l, Z, h) from WAITING
   if ∀(l', Z') ∈ PASSED. l ≠ l' ∨ Z ⊈ Z'
   then add (l, Z) to PASSED
        forall (l', Z', h') s.t. (l, Z, h)⇒(l', Z', h') do
           add (l', Z', h') to WAITING
        od
   fi
od
```

Figure 6.5: Reachability algorithm based on extended zones. The function `select` selects and removes the extended zone from the waiting list with the largest priority.

assign a heuristic value to a symbolic state based on the symbolic state itself and the heuristic value of its predecessor. We call an extended symbolic state $(l', \mathbf{Z}', h')$ a successor of $(l, \mathbf{Z}, h)$, if $(l', \mathbf{Z}')$ is a successor of $(l, \mathbf{Z})$ and $h' = H(l', \mathbf{Z}', h)$. We denote this relation with $(l, \mathbf{Z}, h) \Rightarrow (l', \mathbf{Z}', h')$. The corresponding reachability algorithm is depicted in Table 6.5.

The algorithm starts with an empty PASSED-list and a WAIT-list that contains the initial state. As long as the WAIT-list is not empty the symbolic state with the largest heuristic value is selected. It is checked whether the PASSED-list holds a superset of this state. Subsets of the selected symbolic state will be removed from the PASSED-list, to keep this list short. If the symbolic state passes the inclusion check, its successors are computed and added to the WAIT-list. To realize the search order we change the WAIT-list to a priority list, such that the elements are decreasingly ordered with respect to the heuristic value. In this way heuristic functions can postpone or promote the exploration of a state.

Heuristic search orders as well as depth-first search may lead to starvation. In this case a state is pushed onto the WAIT-list but it is never explored, since newly generated states are added in front of this state to the WAIT-list. Reachability analysis of hybrid automata does not have to terminate, but a wrong search order like depth-first for instance may make things worse. The hybrid automaton in Figure 6.1 illustrates that a certain search order may lead to starvation, while the algorithm terminates for another search order. Breadth-first exploration of the automaton shows within a few iterations that the reachable states in both locations satisfy $0 \leq x \leq 2 + t_{sample}$. If we explore firstly only zones with location loc_1, the exploration can go on forever, since each new zone is slightly bigger than the previous ones. None of the zones will be included in one of its predecessor. Note that

an implementation that uses numerical methods like linear programming is likely to stop eventually, due to numerical errors.

Starvation is a phenomenon which is known, for example, from dynamic scheduling in operating systems. There are scheduling policies that avoid starvation and try to minimize the average waiting time, simultaneously. One assumes that the processing time is proportional to the size of a job. In model checking this is not the case; the processing time – the time needed to compute the successor – of a symbolic state is usually independent of the heuristic value. We consider a strategy which allows an easy implementation, and does not explicitly concern about the average waiting time.

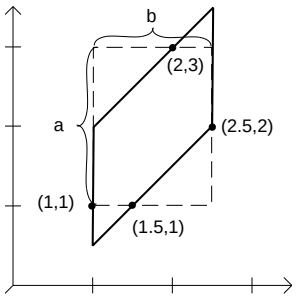
We want that the priority of a state increases linearly with an increasing waiting time, independent of the heuristic value. Let $(l, \mathbf{Z}, h)$ and $(l', \mathbf{Z}', h)$ be the n-th and m-th state that enters the WAIT-list, respectively. We assume that both states have the same heuristic value, and $n < m$. We then want that the first state that entered the WAIT-list has a priority that is $\lambda\,(m-n)$ higher than the priority of the other. It is not necessary to update the priority with each iteration. It is sufficient to assign to each the n-th state, with heuristic value h, priority $p := h - \lambda n$. There is no need to re-order the elements of the WAIT-list. It is sufficient to insert incoming states according to their priority. To avoid starvation, we have to assume additionally that the heuristic function is bounded, i.e. there exist a $h_{max} \in \mathbb{R}$ such that $H(l, \mathbf{Z}, h) \leq h_{max}$. If we choose a big λ the search order becomes more breadth-first like. In contrast, if we choose λ to be 0, the state-space will be explored according to the heuristic function.

Even if example 6.1 suggest otherwise, the efficiency of the algorithm might increase, if one chooses the largest zone rather than choosing breadth-first. This increases the chance that subsequent zones are included and this may reduce the number of iterations needed to explore the complete state-space. The volume of a zone may not be the best way to determine the search order. A reset of a continuous variable for example yields a zone with volume 0. Furthermore, to reduce the overhead we might prefer a measure that is computationally cheap.

As mentioned above, zones are represented by linear inequalities of the form $\mathbf{C}\,x \leq b$, with $\mathbf{C} \in \mathbb{R}^{n \times m}$ and $b \in \mathbb{R}^m$. We refer to the rows of matrix $\mathbf{C}$ with $\mathbf{C}_{(i,\cdot)}$ and to the elements of a vector b with $b_{(i)}$. We then represent the reachable set by matrix $\mathbf{C}$ and a multi-set of $x_1, \ldots, x_m \in \mathbb{R}_n$, that satisfy $\mathbf{C}\,x_i \leq b$ and $\mathbf{C}_{(i,\cdot)}\,x_i = b_{(i)}$. We use these points to compute the successor of a zone. In this chapter we consider the following heuristic functions:

- H_{mean} takes

$$\frac{1}{n} \sum_{i=1,\ldots,n} \left(\max_{j=1,\ldots,m} x_{j_{(i)}} - \min_{j=1,\ldots,m} x_{j_{(i)}} \right)$$



$$\begin{pmatrix} 1 & 0 \\ -1 & 0 \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix} x \leq \begin{pmatrix} 2\frac{1}{2} \\ -1 \\ \frac{1}{2\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}$$

Figure 6.6: A zone $\mathbf{Z}$ is represented by points on the facets and a matrix with the normals on the facets. For the zone depicted if this figure we have $H_{\text{mean}}(\mathbf{Z}) = 0.5\,(a+b) = 1.75$, $H_{\max}(\mathbf{Z}) = \max(a,b) = 2$ and $H_{\Sigma}(\mathbf{Z}) = 0.25\,(1.5 + 0.75\sqrt{2})$.

as heuristic value. There is a smallest cube with edges parallel to the axes, that contains the points x_i. H_{mean} takes the average length of the edges.

- $H_{\max}$ takes

$$\max_{i=1,\ldots,n} \left(\max_{j=1,\ldots,m} x_{j_{(i)}} - \min_{j=1,\ldots,m} x_{j_{(i)}} \right)$$

takes the maximal length of the edges, and

- H_{Σ} uses the sum

$$\frac{1}{m} \sum_{i=1,\ldots,m} \mathbf{C}_{(i,\cdot)} x_i$$

to order the WAIT-list, assumed that the rows vectors $\mathbf{C}_{(i,\cdot)}$ have unit length. Heuristic H_{Σ} takes the mean of the bounds $b_{(i)}$. Figure 6.6 gives an example for this heuristic functions.

Besides heuristics there are other possibilities which may improve the efficiency. An approach that has proven to be beneficial is an additional inclusion check on the WAIT-list. Recall the algorithm in Table 6.5. Before a we add a state to the WAIT-list, it is checked whether the list does not already contain a superset of that state. We add that state only if this is not the case, and remove all symbolic states that are a subsets of the state. This overhead may pay off if it eliminates a sufficient number of symbolic states. In this case we profit from the fact that the WAIT-list is usually shorter than the PASSED-list. We take this as example to investigate how ordering with respect to the size combines with common modifications of the WAIT-list.

6.5 The Case Studies

This section presents the case studies and the verification results. The results in this section were obtained for a MATLAB implementation of the reachability algorithm on a Pentium II 500MHz processor with 256MB memory. The implementation aims to show that polyhedra are suitable for verification; it spends less attention to efficiency. For both examples we use the same basic algorithm, as depicted in Figure 6.5. But we use, as mentioned before, different approaches to compute the successor. We apply the approach that is depicted in Figure 6.3(a) to compute the successor for the LEGO car example. The analysis of the EHC example takes the alternative approach 6.3(b), that uses the same matrix $\mathbf{C}$ to define all zones.

As a consequence, the algorithm uses also different ways to compute the intersection of zones. If we re-use matrix $\mathbf{C}$ it suffices to compare the bounds, and it is not necessary to use any linear programming. For all instances that we consider we have also that the bounds are either monotonic increasing or decreasing on interval $[0, t_{sample}]$. It is therefore sufficient to consider only the bounds at the begin and end of the interval.

The algorithm uses MATLAB's standard Linear Programming module to determine the optimum of linear functions on a polyhedron. This introduces numerical errors. Both problems allow the assumption that these errors remain negligible. The approximation might introduce an error in the range on 10^{-10}. The problems are such that this error does not propagate. Since a symbolic state is at most the $10\ 000$th successor of the initial state, we can assume a numerical error in the order of 10^{-5}. Note however that the zones are over-approximations of the actual zones. This error might propagate if we use the alternative approach of Figure 6.3(b), but we will see that the obtained bounds are nevertheless tight.

6.5.1 The Lego Car

The first case study is taken from a setup that is used at the University of Nijmegen to illustrate the use and need of formals methods to students from secondary school and university. The setup consists of a train, a train gate and a car. There are sensors to detect an approaching train and to detect whether the train gates close properly. The train gate uses an infrared interface for communication with the train in case the gate fails. The car has three sensors, two that allow it to follow a black line and one to detect the state of the traffic light in front of the train gates (Figure 6.7(a)). Jeroen Kratz, who built the setup, verified the correctness of the controller of the train gate with the model checking tool UPPAAL [Kra00].

The car is equipped with a LEGO RCX brick. This brick periodically executes a control program written in NQC, a C like language especially designed for the

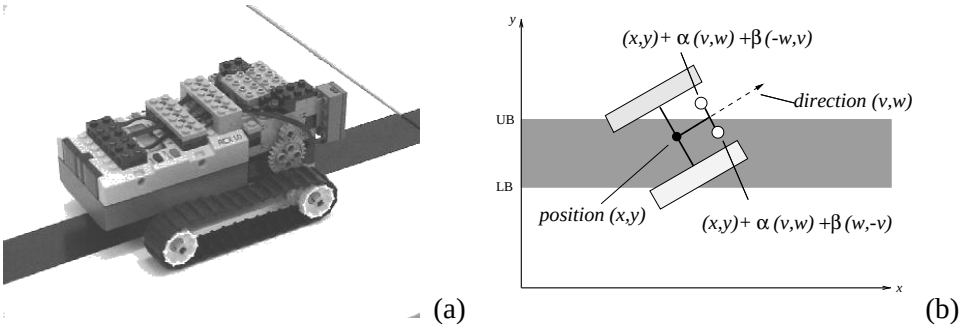


Figure 6.7: The LEGO car is controlled by a single RCX brick. The light sensors for the black line are located between the front wheels. The light sensor at the top in front of the car is used to detect the traffic light at the train gate.

RCX platform. When the car is put on the black line and switched on, it moves forward as long as both sensors detect the black line. If the right sensor detects the white background, the control program reverses the turning direction of the left caterpillar. This results in a turn to the left, while the central position of the car remains unchanged. Similar, if the left sensor detects the white background, the direction of the right caterpillar is reversed. The control program does not take into account explicitly that both sensors detect the white background. But the control task for the left and right sensor can and will be executed concurrently in this case, and the car moves backward.

We model the car by its central position (x, y) and its direction (v, w) (Figure 6.7(b)). If the car moves forward, it has velocity V. The position changes according to the differential equations $\dot{x} = \mathrm{V}\ v$ and $\dot{y} = \mathrm{V}\ w$, while the direction remains unchanged. The positions of the left and right sensors are $(x, y)+\alpha(v, w)+\beta(-w, v)$ and $(x, y) + \alpha(v, w) + \beta(w, -v)$, respectively. The parameters α and β determine the distance between the sensor and the center of the car. The black tape stretches, parallel to the x-axis, between upper bound UB and lower bound LB in the direction of the y-axis. The left sensor detects the tape as long as $\mathrm{LB} \leq y + \alpha\ w + \beta\ v \leq \mathrm{UB}$ holds. If this does not hold, but the right sensor does detect the tape, the car turns to the right. The direction changes according to $\dot{v} = 2\pi\,\omega\,w, \dot{w} = -2\pi\,\omega\,v$, while position (x, y) remains unchanged. The parameter ω gives the number of revolutions per time unit. If both sensor detect the white background the car will move back, and if only the left sensor detects the tape it will move left. This behavior is modeled analogously to moving forward and right.

Table 6.8 shows the hybrid automaton which models the full behavior of the car. Since the black line is assumed to be parallel to the x-axis, we do not need to

actions	`goforward`,`goback`,`goleft`,`goright`
continuous variables	$y, v, w \in \mathbb{R},\ t_{clock} \in \mathbb{R}^{\geq 0}$
discrete variables	$location \in \{\text{FORWARD}, \text{BACK}, \text{LEFT}, \text{RIGHT}\}$
initial condition	$location = \text{FORWARD}, \text{LB} \leq y \leq \text{UB}, v^2 + w^2 = 1, 1/\sqrt{2} \leq v$

discrete transitions

`goforward`:
Pre: $\wedge\ t_{clock} = t_{sample}$
$\wedge\ y + \alpha\, w - \beta\, v \leq \text{UB}$
$\wedge\ y + \alpha\, w - \beta\, v \geq \text{LB}$
$\wedge\ y + \alpha\, w + \beta\, v \leq \text{UB}$
$\wedge\ y + \alpha\, w + \beta\, v \geq \text{LB}$
Eff: $location$:=FORWARD
$t_{clock} := 0$

`goback`:
Pre: $\wedge\ t_{clock} = t_{sample}$
$\wedge \vee\ y + \alpha\, w - \beta\, v \geq \text{UB}$
$\vee\ y + \alpha\, w - \beta\, v \leq \text{LB}$
$\wedge \vee\ y + \alpha\, w + \beta\, v \geq \text{UB}$
$\vee\ y + \alpha\, w + \beta\, v \leq \text{LB}$
Eff: $location$:=BACK
$t_{clock} := 0$

`goleft`:
Pre: $\wedge\ t_{clock} = t_{sample}$
$\wedge \vee\ y + \alpha\, w - \beta\, v \geq \text{UB}$
$\vee\ y + \alpha\, w - \beta\, v \leq \text{LB}$
$\wedge\ \ y + \alpha\, w + \beta\, v \leq \text{UB}$
$\wedge\ \ y + \alpha\, w + \beta\, v \geq \text{LB}$
Eff: $location$:=LEFT
$t_{clock} := 0$

`goright`:
Pre: $\wedge\ t_{clock} = t_{sample}$
$\wedge\ \ y + \alpha\, w - \beta\, v \leq \text{UB}$
$\wedge\ \ y + \alpha\, w - \beta\, v \geq \text{LB}$
$\wedge \vee\ y + \alpha\, w + \beta\, v \geq \text{UB}$
$\vee\ y + \alpha\, w + \beta\, v \leq \text{LB}$
Eff: $location$:=RIGHT
$t_{clock} := 0$

trajectories:
$\dot{t}_{clock} = 1$
$t_{clock} \leq t_{sample}$

If $location = \text{FORWARD}$	then $\dot{y} = \ \text{V}\, w,$	$\dot{v} = \ 0,$	$\dot{w} = \ 0$
If $location = \text{BACK}$	then $\dot{y} = -\text{V}\, w,$	$\dot{v} = \ 0,$	$\dot{w} = \ 0$
If $location = \text{LEFT}$	then $\dot{y} = \ 0,$	$\dot{v} = -2\pi\omega\, w,$	$\dot{w} = \ 2\pi\omega\, v$
If $location = \text{RIGHT}$	then $\dot{y} = \ 0,$	$\dot{v} = \ 2\pi\omega\, w,$	$\dot{w} = -2\pi\omega\, v$

Table 6.8: Hybrid automaton for the LEGO car on a straight line.

include x into the model. We assume that the car and the sensors are initially on the black line and moves forward. The angle between the driving direction and the x-axis lies initially in the interval $[-45, 45]$ degrees.

Verification Results The physical car is designed to drive on a tape with width 2.5 cm. Its speed V is 13 cm/s, and it can make a full turn within 2.5 seconds, thus $\omega = (2.5\text{s})^{-1}$. We assume that the RCX brick has a sampling time t_{sample} of 0.1 seconds. The distance between the sensors is 16 mm, the distance between the center of the sensors to the center of the car is 22 mm, thus $\alpha = 2.2$ cm and $\beta = 0.8$ cm.

We assume that the car and the sensors are initially on the black line and that the car is initially moving forward to the right with an angle between -45 and 45 degrees with the x-axis. This means the set of initial values for (v, w) constitutes

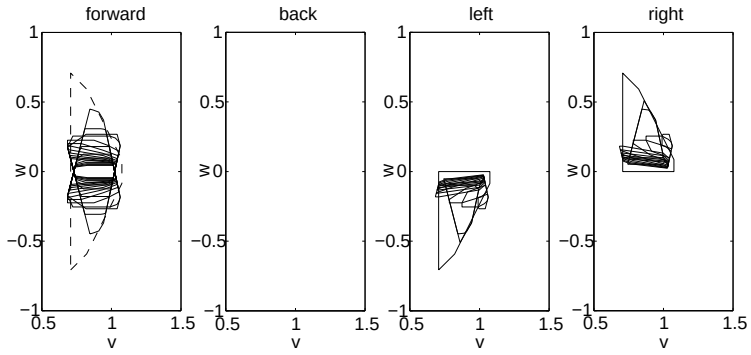


Figure 6.9: Projection of the symbolic states of the LEGO car to v and w. The dashed line in location forward shows the projection of the initial zone.

an arc. The dashed polyhedron in Figure 6.9 shows the polyhedron which was used to over-approximate this set.

We were able to verify that the center of the car never exceeds the upper and lower bound of the tape, and that the car never drives backwards. Analysis of the set of reachable symbolic states yields that the car moves with at least 8.9 cm/s in the direction of the x-axis, if it is in FORWARD mode. The right sensor is never closer to the upper bound than 2.1 mm. Symmetrically, the distance between left sensor and lower bound is also always greater than 2.1 mm. We can therefore conclude that the car shows no unexpected behavior, as long as the car is initially placed on the tape with an angle between the border of the tape and its driving direction of less than 45 degrees. Experiments with the physical car confirm these results.

6.5.2 The Electronic Height Control

This case study deals with an automotive electronic height controller (EHC) that keeps the distance between the chassis and a wheel within bounds. This case study was presented by Stauner, Müller and Fuchs in [SMF97]. We follow their model as closely as possible. For further technical details and a motivation of the specific choices within this model see also [Sta97].

The system consists of different components, as depicted in Figure 6.10. First, we have the chassis, whose height can be changed by pneumatic suspension with a compressor and an escape valve. The measured height passes a low-pass filter, which filters high-frequency disturbances caused for example by holes in the road. The electronic height control (EHC) uses the filtered height to decide whether to use the compressor or the escape valve or to do nothing.

The chassis level is influenced by external disturbances and by the escape valve

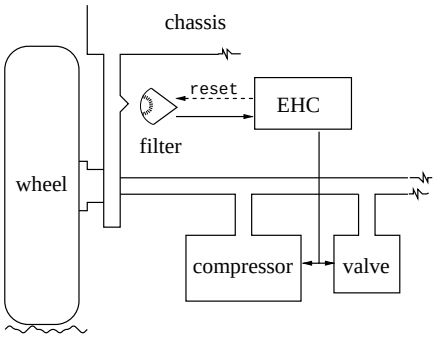


Figure 6.10: The EHC in its environment.

and compressor. The rate of change of the height h of the chassis is the sum of the changes due to disturbances, denoted by e, and the changes due to the compressor and escape valve, denoted by c. The continuous behavior of h is modeled by the linear differential equation.

$$\dot{h} = e + c \tag{6.10}$$

If the controller uses the escape valve, the height h decreases with a rate c in interval $[ev_{min}, ev_{max}]$, while using the compressor increases the height h with $c \in [cp_{min}, cp_{max}]$. To ensure that the disturbances e cannot lead to an unbounded increase or decrease of the height we assume $ev_{max} \leq e \leq cp_{min}$. The filter keeps track of the height, with the restriction that it takes some time until changes in height are properly detected. This feature is useful, because it limits the influence of brief and small disturbances. The filter is modeled by differential equation

$$\dot{f} = \frac{1}{T}(h - t) \tag{6.11}$$

Here the constant T determines the time the filter needs to adjust the filtered height properly. Furthermore, the filter can be reset if necessary. The filtered height will then be set to the set-point, regardless the actual height.

Initially, the controller is in control location IN_TOLERANCE and neither the escape valve nor the compressor are used. Hence, we have $c = 0$. If the filtered height exceeds an upper limit otu, then the controller enters control location DOWN, with the consequence that the height decreases. If the controller is in location DOWN and the filtered height falls below a given upper limit itu, then the controller re-enters control location IN_TOLERANCE and resets the filtered height to the set-point sp. Similarly, there is a control location UP, which is entered if the filtered height f falls below a lower limit otl. In this location we have $c \in [cp_{min}, cp_{max}]$. The controller re-enters IN_TOLERANCE when f exceeds itl. The controller resets the

actions	`stay, to_down, to_up,back`
continuous variables	$c, e, f, h \in \mathbb{R},\ t_{clock} \in \mathbb{R}^{\geq 0}$
discrete variables	$loc \in \{\text{DOWN}, \text{UP}, \text{IN\_TOLERANCE}\}, mode \in \{s, d\}$
initial condition	$t_{clock} = 0 \wedge c = 0 \wedge loc = \text{IN\_TOLERANCE}, h = sp, f = sp$

discrete transitions:

`to_down(`m`):`

Pre: $\wedge t_{clock} = t_{sample}$
$\wedge loc \in \{\text{IN\_TOLERANCE}, \text{UP}\}$
$\wedge (loc = \text{UP}) \rightarrow (m = mode)$
$\wedge f \geq otu_m$

Eff: $loc := \text{DOWN}$
$t_{clock} := 0$
$c :\in [ev_{min}, ev_{max}]$
$mode := m$

`stay:`

Pre: $\wedge t_{clock} = t_{sample}$
$\wedge \vee \wedge loc = \text{IN\_TOLERANCE}$
$\wedge \ \vee f \in [otl_s, otu_s]$
$\vee f \in [otl_d, otu_d]$
$\vee \wedge loc = \text{DOWN}$
$\wedge f \geq itu_{mode}$
$\vee \wedge loc = \text{UP}$
$\wedge f \leq itl_{mode}$

Eff: $t_{clock} := 0$

`to_up(`m`):`

Pre: $\wedge t_{clock} = t_{sample}$
$\wedge loc \in \{\text{IN\_TOLERANCE}, \text{DOWN}\}$
$\wedge (loc = \text{DOWN}) \rightarrow (m = mode)$
$\wedge f \leq otl_m$

Eff: $loc := \text{UP}$
$t_{clock} := 0$
$c :\in [cp_{min}, cp_{max}]$
$mode := m$

`back:`

Pre: $\wedge t_{clock} = t_{sample}$
$\wedge \vee \wedge loc = \text{DOWN}$
$\wedge f \in [otl_{mode}, itu_{mode}]$
$\vee \wedge loc = \text{UP}$
$\wedge f \in [itl_{mode}, otu_{mode}]$

Eff: $loc := \text{IN\_TOLERANCE}$
$t_{clock} := 0$
$c := 0$
$f := sp$

trajectories:

$\dot{t}_{clock} = 1$
$t_{clock} \leq t_{sample}$
$e \in [cp_{max}, ev_{min}]$
$\dot{f} = \frac{1}{T}(h - f)$
$\dot{h} = e + c$

If $loc = \text{IN\_TOLERANCE}$ then $c = 0$
If $loc = \text{UP}$ then $c \in [cp_{min}, cp_{max}]$
If $loc = \text{DOWN}$ then $c \in [ev_{min}, ev_{max}]$

Table 6.11: Hybrid automaton modeling the automotive control problem

filter, to avoid an accumulation of errors in the filter. To get a realistic model, we assume $otl \leq itl \leq sp \leq itu \leq otu$.

If the controller leaves IN_TOLERANCE it makes a nondeterministic choice between the modes *driving* and *stopped*. The hybrid automaton of the EHC (Table 6.11) uses the modes *s* for the stopped car and *d* for the driving car. The controller uses different values for otl, itl, itu, otu, depending on the mode. The thresholds of the driving car are smaller, i.e. closer to the set-point. The model assumes additionally that transitions can only be taken every t_{sample} seconds.

Reachability analysis of the EHC The EHC case study was used in several papers to illustrate hybrid verification. They all derived bounds on the actual height of the chassis with respect to the setpoint. Most authors use a restricted model that

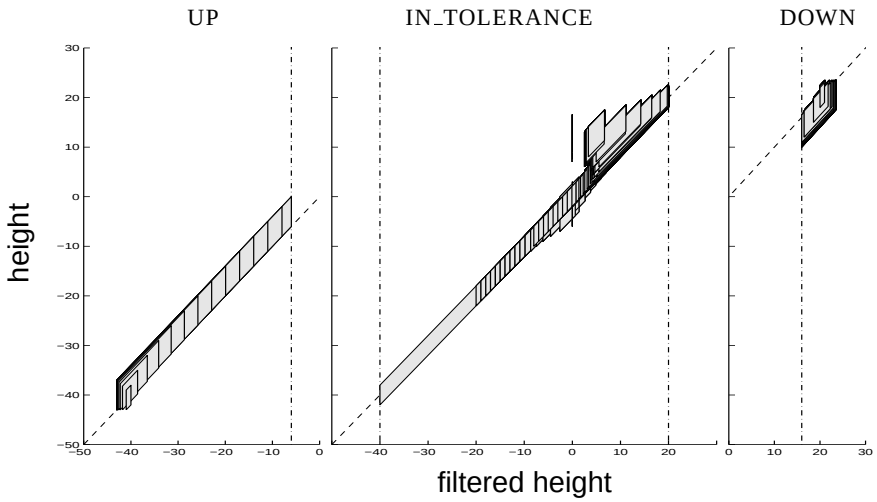


Figure 6.12: The symbolic states of the EHC with $T = 2$, $t_{sample} = 1$ sec and $mode = s$. The dash-dotted lines depict the guards.

considers only the mode *s* of a stopped car. This restriction seems to be justified, since the tolerance limits of the driving mode are tighter. As a consequence the reachable height in the mode *d* is likely to be smaller than in mode *s*. We use for most experiments the restricted model. The results of the full model show that the restriction was justified.

We derived the bounds of the chassis level for a system with $cp_{min} = 1\frac{\text{mm}}{\text{s}}$, $cp_{max} = 2\,\frac{\text{mm}}{\text{s}}$, $ev_{min} = -2\,\frac{\text{mm}}{\text{s}}$, $ev_{max} = -1\,\frac{\text{mm}}{\text{s}}$ and $sp = 0\,\text{mm}$. The outer tolerance limits of the stopped car are defined to be $otl_s = -40\,\text{mm}$, $otu_s = 20\,\text{mm}$. This inner tolerance limits are $itl_s = -6\,\text{mm}$, $itu_s = 16\,\text{mm}$. The corresponding parameters of a driving cars are $otl_d = -10\,\text{mm}$, $otu_d = 10\,\text{mm}$,$itl_d = -6\,\text{mm}$ and $itu_d = 6\,\text{mm}$.

Stauner et al. approximate the system with linear hybrid system, i.e. a hybrid system with piecewise constant derivatives. This method is based on the method presented in [HWT96, HH95]. Stauner et al use as time constant of the filter $T = 2\text{s}$ and as sampling time $t_{sampe} = 1\text{s}$. Using this setting they verify that the chassis level h is always in $[-47\,\text{mm}, 27\,\text{mm}]$. This means that the outer limits otl_s and otu_s are never exceeded by more than $7\,\text{mm}$. They expect that the results can be improved by using a smaller time constant T and a smaller sampling time [Sta97].

We re-examine these results for $T \in \{2\text{s}, 1\text{s}\}$ and $t_{sample} \in \{0.5\text{s}, 1\text{s}\}$. The zones are defined by bounds on the height h, the filtered height f and the difference $f - h$. Matrix $\mathbf{C}$ is consequently $(1\ 0; -1\ 0; 1\ -1;\ -1\ 1; 0\ 1; 0\ -1)$. The bounds on the height for both the restricted and the full model are

	$t_{sample} = 1\text{s}$	$t_{sample} = 0.5\text{s}$
$T = 2\text{s}$	$[-43.00, 23.54]$	$[-42.50, 23.24]$
$T = 1\text{s}$	$[-42.00, 22.10]$	$[-41.50, 21.60]$

These bounds are based on an over-approximation of the reachable states. But the example trace in Table 6.13 shows that these bounds are reasonably tight.

Bemporad and Morari use linear/mixed-integer programming to analyze this case study [BM99]. They derive bounds of $[-44.54\,\text{mm}, 25.00\,\text{mm}]$ for the restricted model with only three control locations, and with $T = 2$ and $t_{sample} = 1$. They use orthogonal polyhedra to approximate reachable states. Elia and Brandin examine the same model in [EB99]. They formulate the reachability problem as a number of linear programming problems, and derive that if the system starts IN_TOLERANCE then $h \in [-43.00, 23.00]$. This result is based on the assumption that the maximum height can be reached with maximal disturbance e. Table 6.13 shows that this assumption does not need to hold if discrete transitions are involved.

Table 6.13: Trace of the EHC with $T = 2$, $t_{sample} = 1\,\text{s}$ and $mode = s$. The controller detects the deviation after 23 seconds. The valve opens and the controller enters IN_TOLERANCE again after another 14 seconds, and resets f to 0.

	Δt	c	e	t	f	h	location
delay	0.1	0	0	0.10	0.00	0.00	IN_TOLERANCE
delay	22.9	0	1	23.00	20.90	22.90	IN_TOLERANCE
transition	-	-	-	23.00	20.90	22.90	DOWN
delay	7.0	-2	1	30.00	17.77	15.90	DOWN
delay	7.0	-1	1	37.00	15.95	15.90	DOWN
transition	-	-	-	37.00	0.00	15.90	IN_TOLERANCE
delay	0.5	0	0	37.50	3.51	15.90	IN_TOLERANCE
delay	6.5	0	1	44.00	19.97	22.40	IN_TOLERANCE
delay	1.0	0	1	45.00	21.15	23.40	IN_TOLERANCE

Step response of the EHC Reachability of a state is the most basic property to verify. One analyzes which states can be reached, assumed that all possible input and disturbances may occur. In Control Theory one also analyzes a systems behavior in reaction to a well defined event. Disturbances of step shape are typical test functions to examine the stability of a controller. In the remainder of this subsection we assume that there is only one disturbance of step shape. At an arbitrary moment, when the system in location IN_TOLERANCE, mode *d* and $f = 0\,\text{mm}$, the height makes a jump to j. We assume additionally that no other disturbances occur, thus $e = 0\,\frac{\text{mm}}{\text{s}}$.

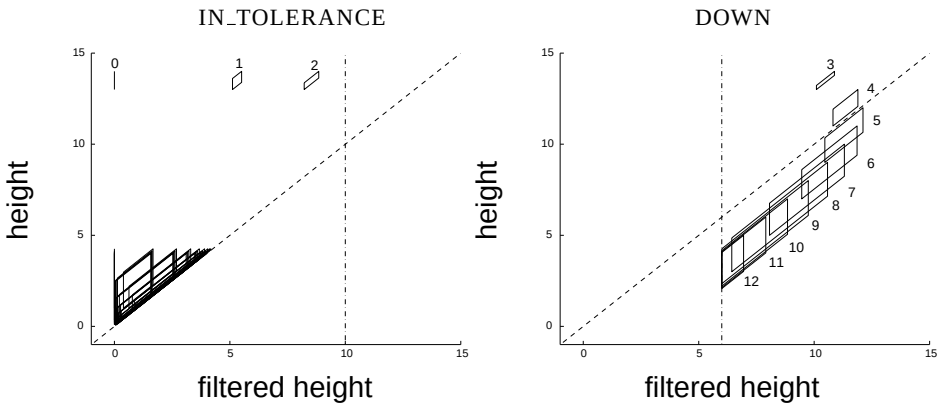


Figure 6.14: Step response due to a jump in $[13\,\mathrm{mm}, 14\,\mathrm{mm}]$ starting at $t_{clock} = 0$.

Stauner et al. assume a jump $j \in (16\mathrm{mm}, 18\mathrm{mm}]$. Additionally they assume that the escape valve operates at its minimum value, hence $ev_{min} = ev_{max} = 1\,\frac{\mathrm{mm}}{\mathrm{s}}$. This restriction was necessary to avoid arithmetic overflows. For this setting they find that the controller leaves IN_TOLERANCE at most $4.3\,\mathrm{s}$ after the disturbance and re-enters it after at most $22.3\,s$. They verify that the chassis level then lies in $[-1\,\mathrm{mm}, 6\,\mathrm{mm}]$.

For the same parameter values we can show a stronger result, namely that the controller leaves location IN_TOLERANCE after at most 3 seconds. The controller re-enters IN_TOLERANCE after at most 16.9 seconds and the chassis level then lies in the interval $[3.0\,\mathrm{mm}, 4.1\,\mathrm{mm}]$. We investigated the step response also for a system with $ev_{min} = -2$ and $ev_{max} = -1$. Figure 6.14 illustrates the behavior of the EHC due to jumps in $[13\,\mathrm{mm}, 14\,\mathrm{mm}]$ occurring at $t_{clock} = 0$. We see that the EHC enters location DOWN after 3 seconds and re-enters location IN_TOLERANCE after at most 13 seconds. All reachable states ultimately converge to points on the diagonal, for the filtered height converges to the real height.

The following table shows for jumps j within the specified intervals how long it takes until the controller detects the disturbance, i.e. enters location DOWN. The escape value opens and the height decreases. The third row gives the time the controller needs at most, to detect the disturbance and to steer the system back to location IN_TOLERANCE. The intervals in the fourth row are the bounds on height when the EHC enters IN_TOLERANCE.

jump j in interval	[11,12]	[12,13]	[13,14]	[14,15]	[15,16]	[16,17]	[17,18]	[18,19]	[19,20]
max. time to detect	5.8	4.6	4.0	3.6	3.2	3.0	2.8	2.7	2.5
max. time to recover	13.8	13.6	14.0	14.6	15.2	16.0	16.8	17.7	18.5
final height h	[0.1,4.4]	[0,4.4]	[0,4.3]	[0,4.4]	[0,4.3]	[0,4.3]	[0,4.2]	[0,4.3]	[0,4.2]

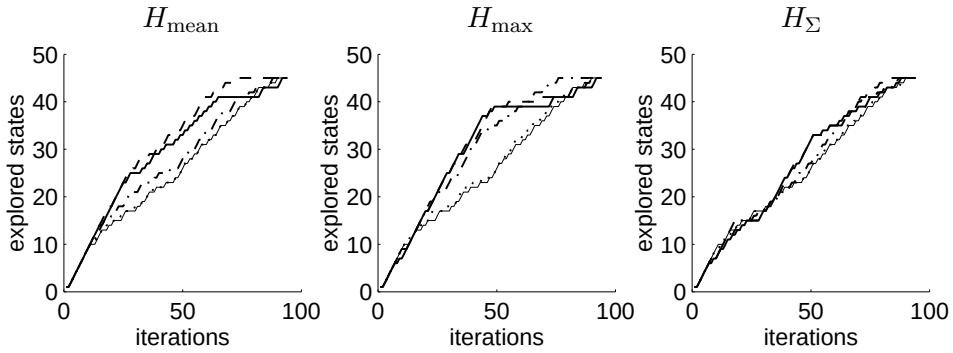


Figure 6.15: Computational results of the LEGO car example. The figures show the number of states that were explored vs. number of iterations. Result for breadth-first (thin solid line), and the heuristic with $\lambda = 0$ (solid), 0.001 (dashed), 0.01 (dashed-dotted), 0.1 (dotted).

6.6 Computational Results

We consider several instances of the two case studies to examine the basic effects of heuristics on the performance of the reachability algorithm. Some heuristics can have positive as well as negative effects. It also depends on the particular instance which effect will eventually prevail.

6.6.1 The Lego Car

As pointed out before, we expect that ordering the WAIT-list with respect to the size of the zones may increase the efficiency. Selecting the biggest zones first can lead to a faster exploration of the state-space. The results for the heuristics H_{mean}, $H_{\max}$ and H_{Σ} confirm this (Figure 6.15). Within less iterations more states are generated compared to the breadth-first search order. The heuristics with $\lambda = 0.1$ behaves similar to breadth-first. The heuristics that sorts the WAIT-list with respect to the size only, are outperformed by the starvation free heuristics with $\lambda = 0.001$.

The results in Figure 6.15 shows that even though the heuristic search orders H_{mean}, $H_{\max}$ and H_{Σ} show a speedup in the beginning, none of them can profit from this advantage. Breadth-first as well as the heuristics need 94 iterations to explore the state-space. Since the heuristics explore the state-space faster in the beginning, they build up the PAST-list faster. A bigger PAST-list has as consequence that the inclusion check becomes more expensive. Each symbolic state has to be compared to the states in the PAST-list. In particular, comparing zones is

expensive, as it involves linear programming. The following table shows for each heuristic the cpu-time and the number comparisons between zones.

	BF	H_{mean}				$H_{\max}$				H_{Σ}			
λ		0.1	0.01	0.001	0	0.1	0.01	0.001	0	0.1	0.01	0.001	0
# comparisons	1192	1195	1265	1372	1396	1217	1330	1370	1365	1198	1207	1209	1226
cpu-time	42.76	42.82	44.07	46.47	47.19	43.58	45.53	46.62	46.38	43.23	43.13	43.37	43.52

It is obvious that ordering the WAIT-list with respect to the priority has in all cases a negative effect on the number of comparisons and thus on the required cpu-time.

This situation changes if we sort not only the WAIT-list but also the past list. If we move states that turned out to be supersets of other states to the head of the list, we may expect a positive effect. This expectation is based on the assumption that these states are more likely supersets of other states, too. If these states are in the beginning of the list, we can expect that it take less comparisons to find a superset of another zone. If we compare a state that is not a subset of any state on the list, then the ordering does not matter. We have to compare it to the full list. Sorting the PAST-list yields the following results.

	BF	H_{mean}				$H_{\max}$				H_{Σ}			
λ		0.1	0.01	0.001	0	0.1	0.01	0.001	0	0.1	0.01	0.001	0
# comparisons	733	733	733	733	733	733	733	733	733	733	733	733	733
cpu-time	31.62	31.76	31.64	31.86	31.95	31.52	31.67	31.56	31.54	31.50	31.51	31.60	31.49

The number of comparisons and hence the cpu-time decreases dramatically, and it does not matter which heuristic was used.

As mentioned before, an additional inclusion check on the WAIT-list is often used to increase the efficiency. The rationale is that the overhead pays off if it reduces the number of waiting states sufficiently. The following table gives besides the cpu-time and the number of comparisons with zones on the past list also the number of comparisons of zones on the wait list.

	BF	H_{mean}				$H_{\max}$				H_{Σ}			
λ		0.1	0.01	0.001	0	0.1	0.01	0.001	0	0.1	0.01	0.001	0
# comp. WAIT	476	490	778	1404	1032	572	1248	1168	1112	494	602	712	580
# comp. PAST	733	733	733	733	733	733	733	733	733	733	733	733	733
cpu-time	42.11	42.91	50.14	70.47	60.02	44.25	63.85	63.06	63.11	42.77	46.18	50.22	47.18

These results show that for the LEGO car example not a single comparison with zones on the PAST-list was saved. As a matter of fact, not a single state was removed from the WAIT-list. Additionally there is of course an overhead by the additional inclusion check. This overhead is larger for a larger WAIT-list. This is a disadvantage for the heuristic search orders, which do not only build up the PAST-list faster, but also the WAIT-list. It must be pointed out that the LEGO car example shows extraordinary behavior. Even though the different heuristics search the state-space in a different order, as shown in Figure 6.15, they all need 94 iterations to explore 45 states and to find 49 states that are included in the PAST-list and not a single explored state is either a sub- or superset of a state on the WAIT-list.

6.6.2 The Electronic Height Control

The first instance of the EHC that we consider has only mode s and $t_{sample} = 1$ and $T = 2$. For this instance we obtain the following results for the reachability analysis without ordering the PAST-list and without inclusion check on the WAIT-list.

	λ	iterations	# comparisons	explored states	cpu-time
BF		2343	925 144	1717	1018.78
H_{mean}	0.1	2059	789 537	1479	859.92
	0.01	1973	636 290	1356	715.46
	0.001	1855	588 495	1278	671.51
	0	1716	620 007	1223	653.18
H_{max}	0.1	2025	670 857	1416	749.62
	0.01	1903	605 087	1302	683.31
	0.001	1893	645 219	1322	697.63
	0	1704	604 820	1214	636.88
H_{Σ}	0.1	2061	765 498	1477	856.82
	0.01	1907	617 030	1314	688.92
	0.001	1845	591 157	1274	651.55
	0	1700	607 325	1211	644.18

In contrast with the previous example ordering the WAIT-list does have a positive effect. The heuristics reduces the number of iterations, explored states, comparisons and the cpu-time. A smaller λ yields in most cases a better result.

Ordering the PAST-list does have a positive effect, too. Moving states that include an active state reduces the number of comparisons, and consequently the cpu-time, as shown by the following table.

	λ	iterations	# comparisons	explored states	cpu-time
BF		2343	878 877	1717	984.74
H_{mean}	0.1	2059	733 162	1479	817.44
	0.01	1973	570 910	1356	665.46
	0.001	1855	534 959	1278	629.95
	0	1716	603 610	1223	642.70
H_{max}	0.1	2025	612 555	1416	704.92
	0.01	1903	546 789	1302	638.94
	0.001	1893	598 911	1322	664.94
	0	1704	589 374	1214	631.36
H_{Σ}	0.1	2061	706 494	1477	796.98
	0.01	1907	553 255	1314	640.75
	0.001	1845	538 400	1274	614.51
	0	1700	591 805	1211	633.51

The effect of this modification is not as dramatic as for the LEGO example. The number of comparisons decreases slightly, compared to the results without ordering the PAST-list. Note that the least number of comparisons is attained for heuristics with $\lambda = 0.001$ or $\lambda = 0.01$, even though they explore more states and iterate more often than heuristics with $\lambda = 0$. The number of iterations, the number of explored states and the number of comparisons are only loosely related, but they all are positively related with the cpu-time.

The additional inclusion check on the WAIT-list pays off, if it deletes a sufficient number of states. For the LEGO car example this was clearly not the case. The additional inclusion check was pure overhead. The results of the EHC show that pruning states from the WAIT-list can improve the efficiency. The number of iterations, explored states and the cpu-time decreases in all cases, the number of comparisons decreases in most cases.

	λ	iterations	# comp. WAIT	# comp. PAST	explored states	cpu-time
BF		1312	84 935	537 298	1114	620.91
H_{mean}	0.1	1348	90 550	522 571	1144	624.53
	0.01	1289	62 697	403 652	1071	500.28
	0.001	1330	37 113	416 639	1057	484.14
	0	1556	29 487	539 178	1168	569.02
H_{max}	0.1	1374	84 857	476 196	1142	563.34
	0.01	1402	40 324	466 113	1122	511.85
	0.001	1422	26 982	466 787	1080	495.96
	0	1544	30 089	525 482	1159	549.35
H_{Σ}	0.1	1384	97 833	51 4464	1176	607.08
	0.01	1322	56 919	413 006	1089	483.41
	0.001	1382	25 093	450 867	1060	479.73
	0	1539	29 904	527 898	1156	552.73

Heuristic search orders reduce the cpu-time with up to 20 %. This does not justify running the algorithm several times to find an optimal heuristic. We apply

the best heuristics of the previous paragraph, H_Σ with $\lambda = 0.01$ and $\lambda = 0.001$, to other instances of the EHC, to see whether the results that were obtained in previous paragraph extend to other instances.

			iterations			explored states			cpu-time		
t_{sample}	T	# modes	BF	H_Σ $\lambda = 0.01$	H_Σ $\lambda = 0.001$	BF	H_Σ $\lambda = 0.01$	H_Σ $\lambda = 0.001$	BF	H_Σ $\lambda = 0.01$	H_Σ $\lambda = 0.001$
2	0.5	1	8329	8166	7497	7571	7005	6418	16969.0	16304.4	14264.9
1	1	1	226	252	244	182	204	196	30.3	31.7	30.8
1	0.5	1	593	612	579	520	544	515	146.7	138.4	124.7
2	1	2	8588	7816	9866	6727	6585	7846	10818.3	9949.0	10049.6
2	0.5	2	56229	55054	61842	48214	48208	54037	418504.4	423446.7	443195.2
1	1	2	1588	1179	1295	1157	913	949	380.3	300.4	294.9
1	0.5	2	5508	3596	3908	4106	3049	3336	3159.2	2416.4	2434.9

If we compare the results of the heuristic search orders, with the results of breadth-first, we see as well an increase as a decrease in the number of iterations and explored states. Still, in most cases we see a decrease of the cpu-time with up to 25%, except for two cases with a slight increase up to about 5%.

The inclusion check on the WAIT-list can effectively reduce the number of comparisons, and thus the cpu-time. Nevertheless, the LEGO car example demonstrated that the overhead of this inclusion check does not need to pay off. Re-ordering the PAST-list in contrast is the only modification that showed a positive effect for that case study. Since this modification is only based on a notion of superset, it can be incorporated in any type of forward reachability algorithm.

Heuristic search orders have in many cases a positive effect, too. But also in this case the LEGO car showed to be resistant to efforts to improve the search order. The results for the EHC show that ordering the WAIT-list with respect to the size can improve the results. But, this ordering can also lead to starvation. We suggested a simple modification, which takes also into account the waiting time of a symbolic state. It turns out that heuristics, based on weighted sum of the size and the waiting time of a state, can outperform search orders that are based on the size of a state only or, like breadth-first, on the waiting time only.

6.7 Conclusion

The computational results show that it clearly matters in which order the states are searched, even if we explore the full state-space. But if we want to explore the full state-space, we would prefer an heuristic that is applicable in many cases, rather than a heuristic that is tailored to the problem. This in contrast to the case, when

we want to find an (error) trace to a particular state. A heuristic that takes the size of a symbolic state and the waiting time into account seems to be a good candidate for an improved state-space exploration, even if the LEGO car example turned out to be stubborn.

We consider only two case studies, which is certainly not sufficient to draw conclusions that are applicable to all cases. They indicate however that search orders other than breadth-first can improve the performance. Other criteria than size of a zone may turn out to be useful as well. The results show however that other modifications can reduce the influence of heuristic search orders.

Search orders that depend on the size of the zone of a symbolic state are not restricted to a certain representation of zones. Other representations that use for example orthogonal polyhedra, ellipsoids, projections or others allow search orders that are based on the size of the zones, too. This extends to model checkers for linear hybrid automata like HyTech [HHWT95], rectangular automata [PKHWT98] or timed automata like KRONOS [Yov97] or UPPAAL [LPY97].

The verification results show that the proposed approximation technique gives tighter bound on the reachable states than those obtained in [SMF97] and [BM99]. It allows us to consider hybrid systems with continuous behavior, that is defined by a linear system with bounded and uncertain input. It uses the full dynamics of the continuous behavior, but needs restrictions on the discrete dynamics. The most restrictive assumption is that the system has to be clocked, which excludes autonomous switching of the dynamics by, for example, collisions. However, many real-life problems within a controller-environment setting have this property, as illustrated by the LEGO car example.

The main disadvantage of the model checking approach is illustrated by the computational results of EHC. By introducing one extra mode, the algorithm has to explore up to 10 times more states. This problem can be solved partially, if one solves the problem of how to take the (non-convex) union of symbolic states. A promising approach based on orthogonal polyhedra was put forward in [Dan99]. The verification results for the EHC show also the advantage of the model checking approach. When analyzing the behavior of a system by hand instead, a supposedly true assumption is made easily, as shown by the example trace in Table 6.13.

The LEGO car example itself is, to our opinion, a nice contribution of this work. In spite of the fact that it is a rather small system, with only four locations and less than 50 symbolic states, it shows interesting non-trivial hybrid behavior. This example can be scaled up easily, for example by considering its behavior in bends, if it is going downhill and starts slipping, or if the two caterpillars do not have the same speed. It shows that hybrid automata are suitable to analyze real life problems even if it is in this case literally a toy example.

7

Conclusions

This is the last chapter of this thesis, and despite of each of the foregoing chapters having its own conclusion, this chapter briefly summarizes what has been achieved, gives some general comments on conducting experiments, and sketches directions for future research.

7.1 On this Thesis

This thesis shows in Chapter 2 how to model planning and scheduling problems as network of timed automata. The Sidmar steel plant model with up to 28 processes and 26 clocks was one of the largest timed automata networks, at that time. To get the required answers for this model we had to modify both the timed automaton framework and the model checking algorithm.

The problems is not to verify whether a state is reachable, but to compute the optimal trace to a goal state. Chapter 3 proposes *Linearly Priced Timed Automata* to capture cost that are attached to delays and transitions. The basic notion of a *Priced Region* then allows to obtain an algorithm that computes the optimal solution. We show that this algorithm terminates for any LPTA. This algorithm however is guaranteed to be fairly inefficient, since it is based on an extension of regions.

The extension of zones, which are used by current model checking algorithm to represent sets of states, to priced zones leads to more efficiency. Chapter 4 presents for the sub-class of *Uniformly Priced Timed Automata*, which allows modeling of minimal time scheduling problems, an algorithm that is essentially as efficient as the model checking algorithm for timed automata, with the difference that it yields the optimal trace to the goal state and not just the first trace that it encounters.

Chapter 5 introduces priced zones for the full class of Linearly Priced Timed Automata. The necessary operations on zones require the use of linear programming in the implementation. Results with a prototype indicate a decrease of efficiency when it comes to minimal time problems, but experiments show clearly

that this approach can compete with other approaches when it comes to scheduling problems that lie outside the restricted class of minimal time problems.

Linearly Priced Timed Automata allow to assign costs to delays and transitions. Data structures based on priced zones then allow to represent sets of states efficiently. But the problem remains that the timed automaton models, and thus the state spaces, tend to be large. Guiding the state-space exploration proved to be an effective remedy to this problem. Chapter 4 presents a modified algorithm that allows to define heuristics search orders.

Chapter 6 is based on earlier work on how to over-approximate the reachable states of hybrid systems. It shows for two case studies from a restricted class of hybrid systems that approximation with polyhedra can lead to tight bounds on the set of reachable states. As a complement to the experiments in the foregoing chapters, it investigates how heuristic search orders can improve the efficiency of the exploration of a complete state-space.

7.2 On Experiments

The work that is presented in this thesis led to prototype implementations that have been applied to a number of case studies and examples. When it comes to comparison with other approaches one faces the problem that case studies from literature do in many cases not provide sufficient information to redo the modeling. Even if there is such information, different models of the same problem tend to differ in details, which may and often does influence the results significantly. For example, the model of the Sidmar steel plant in [Feh99] and the model in Chapter 2 differ in many aspects, mainly because ambiguities in the informal description have been clarified in the meanwhile.

Benchmarks can serve to circumvent the problems of ambiguous informal problem descriptions. A prerequisite for a good benchmark is a precise, ideally formal, description of the problem. The instances of the job shop problem and the aircraft landing problems (Chapter 4) are typical benchmarks; each instance can be effectively represented as a single matrix. The Electronic Height Control serves also as a benchmark, given the formal definition in [SMF97]. But not any formal model constitutes a good benchmark; it should be free from details that derive from a particular approach or tool. The UPPAAL models of job shop problems as presented in Section 4.5.3, for example, use urgent channels, a concept that is not present in other timed automaton model checkers. Even though the models are precise and formal, they are not suitable for comparison of different tools. This is in contrast with generic job shop problems, which can be modeled in any timed automaton framework.

If we want to compare the efficiency of an approach, benchmarks are often not sufficient. The are numerous results for the job shop problem, but most of them are unfortunately outdated. The performance of a modern computers exceeds the performance of those that were used to obtain the results in the literature by several orders of magnitude. Experiments with different approaches on the same computer are in practice frustrated by the fact, that the executable or source codes are only in a few cases available.

The LINPACK benchmark [Don01] provides a way to compare the performance of different computers, but when running the benchmark on modern PCs, one has to face the problem that the execution time of the benchmark might be smaller than the granularity of the timer. Only a modification of the function that is used to time the cpu time allowed us to benchmark the Pentium II 330MHz that is used in Chapter 5. These results have been in included in recent versions of [Don01].

Even if we have a precise problem description and the computers are comparable, it remains that little differences between models, even when they have no semantical implications, may lead to a different outcomes. Changing the position of two automata in the system description for example, may turn a tractable problem into a intractable one. In contrast with other experimental sciences, we cannot lean on the principle of strong causality, namely that small causes should have small consequences. Rerunning the same experiment several times does not help, since this kind of coincidence is reproducible. Guiding the state-space exploration helps to soften this problem for the model checking experiments that were conducted for this thesis, since it restricts the possible evaluation orders.

Experiments in this thesis, and in Computer Science in general, can be used for three different purposes. First, there are experiments that should prove the concept. The problem to be solved serves often only to illustrate a new methodology. These experiments give the experimenter the freedom to include interesting aspects into the problem. The bridge problem in this thesis is a typical example, since it is used to illustrates several extensions of the model checking algorithm.

Secondly, this thesis contains a number of experiments related to case studies. These serve to show the applicability of an approach in practice. These experiments might solve the proposed problem, but they also help the researcher to learn about the needs of the field, and vice versa, the field learns about what methodologies are available and applicable. Finally, we have experiments on benchmarks that allow to compare different approaches. Libraries of benchmarks are extremely useful for this kind of research, and fortunately there are some good libraries with optimization benchmarks available online. It would certainly ease doing experimental research in formal methods, if similar libraries with verification problems would be available, too.

7.3 On Future Research

An obvious first direction of future research is to work on stable implementations of the prototypes that have been developed for LPTAs and UPTAs. The syntax that is accepted by the current prototypes is very restricted, such that modeling costs and heuristics can become cumbersome. It allows on the other hand to violate basic sanity properties, like that the sum of the cost and the remaining cost should not decrease. Future work on these issue is necessary to come to an implementation that can be distributed, either as integrated part of UPPAAL or as a separate cost-optimizing version.

We identified the class of job shop problems as a larger class that fits in this framework of LPTA. The same holds for similar problems with as objective to minimize the tardiness or earliness, like the aircraft landing problem. But for most other scheduling or planning problems, the only way to show that they can be modeled as LPTA, is actually to model them as LPTA. Future research should focus on identifying bigger classes of scheduling problems that are covered by LPTA; if possible it should aim at giving sufficient and necessary conditions for a problem to be in the class of problems that can be modeled by a LPTA.

Heuristics search orders have proven to be a suitable way to reduce the searched state-space. In this thesis most heuristic orders are tailored to a specific problem. It would be valuable to identify heuristics that can be applied to several problems. This thesis presents some first results on minimal-cost heuristics in Chapter 4, and on heuristics that depend on the size of a zone in Chapter 6. Partial order reduction of untimed systems has proven to be a powerful mechanism to reduce the state-space, but the results, unfortunately, do not extend easily to timed and hybrid systems. A promising direction of future research would be to investigate whether a kind of ”light-weight” partial order reduction can be obtained by heuristic search orders.

Model checking technology has proven over the last decades to be suitable for verification of hard- and software systems. In this thesis we have shown that this technology can be successfully applied in the domain of scheduling and planning. As a next step future research should focus on extending the application domain of model checking techniques further; a direction that has recently been suggested is to use model checking for the generation of test cases.

Bibliography

[ABB+01] T. Amnell, G. Behrmann, J. Bengtsson, P.R. D'Argenio, A. David, A. Fehnker, T.S. Hune, B. Jeannet, K.G. Larsen, M.O. Möller, P. Pettersson, C. Weise, and W. Yi, *UPPAAL - Now, Next, and Future*, Modeling and Verification of Parallel Processes (MOVEP'2k) (F. Cassez, C. Jard, B. Rozoy, and M. Ryan, eds.), LNCS 2067, 2001, pp. 100–125. 18

[AC91] D. Applegate and W. Cook, *A Computational Study of the Job-Shop Scheduling Problem*, OSRA Journal on Computing 3 (1991), 149–156. 20, 21, 28, 85

[ACH+95] R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine, *The algorithmic analysis of hybrid systems*, Theoretical Computer Science **138** (1995), 3–34. 14, 18

[ACH97] R. Alur, C. Courcoubetis, and T. A. Henzinger, *Computing accumulated delays in real-time systems*, Formal Methods in System Design: An International Journal **11** (1997), no. 2, 137–155. 43

[AD94] R. Alur and D.L. Dill, *A Theory of Timed Automata*, Theoretical Computer Science **126** (1994), 183–235. 14, 18, 20, 30, 45, 47

[AM99] E. Asarin and O. Maler, *As soon as possible: Time optimal control for timed automata*, Hybrid Systems: Computation and Control (F.W. Vaandrager and J.H. van Schuppen, eds.), LNCS 1569, Springer, 1999, pp. 19–30. 17, 43

[AM01] Y. Abdeddaïm and O. Maler, *Job-Shop Scheduling using Timed Automata*, 13th Conference on Computer Aided Verification (CAV'01), LNCS 2102, 2001. 17, 85

[AN00] P.A. Abdulla and A. Nylén, *Better is better than well: On efficient verification of infinite-state systems*, LICS00, IEEE, 2000. 66

[ATP01] R. Alur, S. La Torre, and G. Pappas, *Optimal Paths in Weighted Timed Automata* , 4th International Workshop on Hybrid Systems: Computation and Control (HSCC'2001), LNCS 2034, 2001. 17, 43

[AvLLU94] E.H.L. Aarts, P.J.M. van Laarhoven, J.K. Lenstra, and N.L.J. Ulder, *A Computational Study of Local Search Algorithms for Job-Shop Scheduling*, OSRA Journal on Computing **6** (1994), no. 2, 118–125. 20, 21, 28

[BFH+01a] G. Behrmann, A. Fehnker, T.S. Hune, K.G. Larsen, P. Petterson, and J.M.T. Romijn, *Efficient Guiding Towards Cost-Optimality in* UPPAAL, TACAS'2001, LNCS 2031, 2001. 18, 43, 67, 108

[BFH+01b] G. Behrmann, A. Fehnker, T.S. Hune, K.G. Larsen, P. Petterson, J.M.T. Romijn, and F.W. Vaandrager, *Minimum-Cost Reachability for Linearly Priced Timed Automata*, 4th International Workshop on Hybrid Systems: Computation and Control (HSCC'2001), LNCS 2034, 2001. 18

[BGK+96] J. Bengtsson, W.O.D. Griffioen, K.J. Kristoffersen, K.G. Larsen, F. Larsson, P. Pettersson, and W. Yi, *Verification of an Audio Protocol with Bus Collision Using* UPPAAL, CAV96 (R. Alur and T.A. Henzinger, eds.), LNCS 1102, Springer–Verlag, 1996, pp. 244–256. 107

[BJLY98] J. Bengtsson, B. Jonsson, J. Lilius, and W. Yi, *Partial order reductions for timed systems*, CONCUR'98 (Vancouver, Canada), LNCS 1427, Springer, June 1998, pp. 485–500. 22, 27

[BJS95] P. Brucker, B. Jurisch, and B. Sievers, *Code of a Branch & Bound Algorithm for the Job Shop Problem*, 1995, Available at url http://www.mathematik.uni-osnabrueck.de/research/OR/. 85

[BKA00] J.E. Beasley, M. Krishnamoorthy, and D. Abramson, *Scheduling Aircraft Landings-The Static Case*, Transportation Science **34** (2000), no. 2, 180–197. 18, 96, 105, 106

[BM99] A. Bemporad and M. Morari, *Verification of hybrid systems via mathematical programming*, Hybrid Systems: Computation and Control (F.W. Vaandrager and J. van Schuppen, eds.), LNCS 1569, Springer Verlag, 1999, pp. 31–45. 129, 136

[BM00] E. Brinksma and A. Mader, *Verification and Optimization of a PLC Control Schedule*, Proceedings of the 7th SPIN Workshop, LNCS 1885, Springer Verlag, 2000. 17, 90, 91

[BMF02] E. Brinksma, A. Mader, and A. Fehnker., *Verification and optimization of a PLC control schedule*, International Journal on Software Tools For Technology Transfer (STTT), Springer Verlag, 2002. 18, 67, 91

[CK99] A. Chutinan and B.H. Krogh, *Verification of polyhedral-invariant hybrid automata using polygonal flow pipe approximations*, Hybrid Systems: Computation and Control (F.W. Vaandrager and J.H. van Schuppen, eds.), LNCS 1569, Springer, 1999, pp. 76–90. 18, 109

[BS99] R. Boel and G. Stremersch, *VHS Case Study 5: Timed Petri net model of steel plant at SIDMAR*, Tech. report, SYSTeMS Group, Universiteit Gent, Technologiepark-Zwijnaarde 9, B-9052 Ghent, Belgium, 1999. 17, 31

[CP89] J. Carlier and E. Pinson, *An Algorithm for Solving the Job-Shop Problem*, Management Science **35** (1989), no. 2, 164–176. 19, 21, 28

[CW96] E.M. Clarke and J.M. Wing, *Formal methods: State of the art and future directions*, ACM Computing Surveys, December 1996, pp. 28(4):626–643. 12, 18

[Dan99] Tao Dang, *Vérification et synthèse des systémes hybrides*, Ph.D. thesis, Verimag, Grenoble, 1999. 18, 136

[DFMV98] H. Dierks, A. Fehnker, A. Mader, and F.W. Vaandrager, *Operational and Logical Semantics for Polling Real-Time Systems*, Proceedings FTRTFT'98 (A.P. Ravn and H. Rische, eds.), LNCS 1486, Springer, 1998, pp. 29–40. 18

[Dil89] D. Dill, *Timing Assumptions and Verification of Finite-State Concurrent Systems*, Proc. of Automatic Verification Methods for Finite State Systems (J. Sifakis, ed.), LNCS 407, Springer–Verlag, 1989, pp. 197–212. 68, 69, 95

[DM98] T. Dang and O. Maler, *Reachability analysis via face lifting*, Hybrid Systems: Computation and Control (T.A. Henzinger and S. Sastry, eds.), LNCS 1386, Springer, April 1998, pp. 96–109. 109

[Don01] J. Dongarra, *Performance of Various Computers Using Standard Linear Equations Software*, Tech. Report CS-89-85, Computer Science Department, University of Tennessee, 2001, An up-to-date version of this report can be found at http://www.netlib.org/benchmark/performance.ps. 106, 139

[DSW98] A. Dolzman, T. Sturm, and V. Weispfennig, *Real Quantifier Elimination in Practice*, School on Computational Aspects and Applications of Hybrid Systems, 1998, Proc KIT Workshop on Verifications of Hybrid systems. 21

[EB99] N. Elia and B. Brandin, *Verification of an automotive active leveler*, Proc. of the 1999 American Control Conference (ACC), 1999. 129

[Feh98] A. Fehnker, *Automotive Control Revisited – Linear Inequalities as Approximation of Reachable Sets*, Hybrid Systems: Computation and Control (T.A. Henzinger and S. Sastry, eds.), LNCS 1386, Springer, April 1998, pp. 110–126. 18, 109, 110

[Feh99] A. Fehnker, *Scheduling a Steel Plant with Timed Automata*, Sixth International Conference on Real-Time Computing Systems and Applications (RTCSA'99), IEEE Computer Society Press, 1999. 18, 19, 31, 39, 138

[Feh00a] A. Fehnker, *Bounding and Heuristics in forward reachability algorithms*, Tech. Report CSI-R0002, Computing Science Institute Nijmengen, 2000. 18, 19, 77

[Feh00b] A. Fehnker, *Heuristic Reachability Analysis of Hybrid Systems*, Unpublished Manuscript, November 2000, Available at http://www.cs.kun.nl/~ansgar/papers/lego.ps.gz. 18, 109

[Fre82] S. French, *Sequencing and scheduling: An introduction to the mathematics of the job-shop*, Mathematics and its Application, Ellis Horwood, Chichester, England, 1982. 18

[GJ79] M. Garey and D.S. Johnson, *Computers and Intractability, A guide to the Theory of NP-Completeness*, W.H. Freeman and Company, San Francisco, 1979. 20

[GM99] M.R. Greenstreet and I. Mitchell, *Reachability analysis using polygonal projections*, Hybrid Systems: Computation and Control (F.W. Vaandrager and J. van Schuppen, eds.), LNCS 1569, Springer, 1999, pp. 103–116. 109

[Hen96] T.A. Henzinger, *The theory of hybrid automata*, Proceedings of the 11th Annual Symposium on Logic in Computer Science, IEEE Computer Society Press, 1996, Invited tutorial, pp. 278–292. 45, 112

[HH95] T.A. Henzinger and P.-H. Ho, *Algorithmic analysis of nonlinear hybrid systems*, CAV 95: Computer-aided Verification (P. Wolper, ed.), LNCS 939, Springer, 1995, pp. 225–238. 110, 128

[HHWT95] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi, *A user guide to* HYTECH, Proceedings of TACAS'95 (E. Brinksma, W.R. Cleaveland, K.G. Larsen, T. Margaria, and B. Steffen, eds.), LNCS 1019, Springer, 1995, pp. 41–71. 109, 136

[Hig52] G. Higman, *Ordering by divisibility in abstract algebras*, Proc. of the London Math. Soc. **2** (1952), 326–336. 65, 104

[HKPV95] T.A. Henzinger, P.W. Kopke, A. Puri, and P. Varaiya, *What's decidable about hybrid automata?*, Proceedings of the 27th Annual Symposium on Theory of Computing, ACM Press, 1995, pp. 373–382. 14, 110

[HLP00] T.S. Hune, K.G. Larsen, and P. Pettersson, *Guided Synthesis of Control Programs Using* UPPAAL, Proc. of the IEEE ICDCS International Workshop on Distributed Systems Verification and Validation (Ten H. Lai, ed.), IEEE Computer Society Press, April 2000, pp. E15–E22. 77, 86

[Hol97] G. Holzmann, *The model checker spin*, IEEE Trans. on Software Engineering (1997), 23:279–295. 91

[HWT96] T.A. Henzinger and H. Wong-Toi, *Linear phase-portrait approximations for nonlinear hybrid systems*, Hybrid Systems III (R. Alur, T.A. Henzinger, and E.D. Sontag, eds.), LNCS 1066, Springer, 1996, pp. 377–388. 128

[ID93] C.N. Ip and D.L. Dill, *Better verification through symmetry*, Computer Hardware Description Languages and their Applications (D. Agnew, L. Claesen, and R. Camposano, eds.), Elsevier Science Publishers B.V., Amsterdam, Netherland, 1993, pp. 87–100. 32

[JM99] A.S. Jain and S. Meeran, *Deterministic job-shop scheduling; past, present and future*, European Journal of Operational Research (1999), volume 113, issue 2. 18, 20

[Kra00] J. Kratz, LEGO *bij Informatica voor Technische Toepassingen*, Computing Science Institute, University of Nijmengen, 2000, Available at http://www.cs.kun.nl/ita/voorlichting/report.ps.gz. 110, 122

[KV00] A. Kurzhanski and P. Varaiya, *Ellipsoidal techniques for reachability analysis*, Hybrid Systems: Computation and Control (N. Lynch and B. Krogh, eds.), LNCS 1790, Springer, 2000, pp. 203–213. 109

[LBB+01] K.G. Larsen, G. Behrmann, E. Brinksma, A. Fehnker, T.S. Hune, P. Petterson, and J.M.T. Romijn, *As Cheap as Possible: Efficient Cost-Optimal Reachability for Priced Timed Automata*, 13th Conference on Computer Aided Verification (CAV'01), LNCS 2102, 2001. 18, 95

[Lio96] J.L. Lions, *Ariane 5 flight 501 failure: Report of the inquiry board*, Available at http://www.esa.int, July 1996. 11

[LLPY97] F. Larsson, K.G. Larsen, P. Pettersson, and W. Yi, *Efficient Verification of Real-Time Systems: Compact Data Structures and State-Space Reduction*, RTSS97, IEEE, 1997, pp. 14–24. 104

[LM67] E.B. Lee and L. Markus, *Foundations of optimal control theory*, The SIAM series in applied mathematics, Wiley, New York, 1967. 114

[LPY95] K.G. Larsen, P. Pettersson, and W. Yi, *Diagnostic Model-Checking for Real-Time Systems*, Proc. of Workshop on Verification and Control of Hybrid Systems III, LNCS 1066, Springer–Verlag, 1995, pp. 575–586. 67

[LPY97] K.G. Larsen, P. Pettersson, and W. Yi, UPPAAL *in a Nutshell*, Int. Journal on Software Tools for Technology Transfer **1** (1997), no. 1–2, 134–152. 18, 22, 23, 31, 104, 136

[LPY99] G. Lafferriere, G.J.. Pappas, and S. Yovine, *A new class of decidable hybrid systems*, Hybrid Systems: Computation and Control (F.W. Vaandrager and J. van Schuppen, eds.), LNCS 1569, Springer, 1999, pp. 103–116. 110

[LSV01] N.A. Lynch, R. Segala, and F.W. Vaandrager, *Hybrid I/O Automata Revisited*, 4th International Workshop on Hybrid Systems: Computation and Control (HSCC'2001), LNCS 2034, 2001. 112

[Min99] M. Minea, *Partial order reduction for model checking of timed automata*, Proceedings CONCUR 99 (Eindhoven, The Netherlands) (J.C.M. Baeten and S. Mauw, eds.), LNCS 1664, Springer-Verlag, 1999, pp. 431–446. 27

[Mit70] L.G. Mitten, *Branch-and-bound Methods: General Formulation and Properties*, Operations Research **18** (1970), 24–34. 40

[NTY00] P. Niebert, S. Tripakis, and S. Yovine, *Minimum-time reachability for timed automata*, IEEE Mediterranean Control Conference, 2000, accepted for publication. 17, 43, 44, 67

[NY99] P. Niebert and S. Yovine, *Computing optimal operation schemes for multi batch operation of chemical plants*, VHS deliverable, May 1999, Available at `http://www-verimag.imag.fr/VHS/year1/cs11e.ps`. 17, 93

[PKHWT98] J. Preußig, S. Kowalewski, T.A. Henzinger, and H. Wong-Toi, *An algorithm for the approximate analysis of simple rectangular automata*, 5th Int. School and Symposium on Formal Techniques in Fault Tolerant and Real Time Systems, LNCS 1486, Springer, 1998, pp. 228–240. 45, 136

[RB98] T. Ruys and E. Brinksma, *Experience with Literate Programming in the Modeling and Validation of Systems*, Proceedings of TACAS'98 (Lisbon, Portugal) (B. Steffen, ed.), LNCS 1384, Springer, April 1998, pp. 393–408. 81

[RE99] F. Reffel and S. Edelkamp, *Error detection with directed symbolic model checking*, FM'99 - Formal Methods (J.M. Wing, J. Woodcock, and J. Davies, eds.), LNCS 1708, 1999, pp. 195–211. 17, 77

[Rok93] T.G. Rokicki, *Representing and Modeling Digital Circuits*, Ph.D. thesis, Stanford University, 1993. 71, 104

[She99] G. Shedler, *Regenerative Stochastic Simulation*, Academic Press, 1999. 91

[SMF97] T. Stauner, O. Müller, and M. Fuchs, *Using HyTech to verify an automotive control system*, HART'97 (O. Maler, ed.), LNCS 1201, Springer, 1997, pp. 139–153. 110, 125, 136, 138

[Sta97] T. Stauner, Specification and Verification of an Electronic Height Control System using Hybrid Automata, Master's thesis, Munich University of Technology, 1997. 125, 128

[Sto00] M. Stobbe, *Results on scheduling the Sidmar steel plant using constraint programming*, 2000, Available at http://www-verimag.imag.fr/VHS/CS5/stobbe.ps.gz. 86

[Tri90] H.W.J.M. Trienekens, *Parallel Branch and Bound Algorithms*, Ph.D. thesis, Erasmus Universiteit Rotterdam, 1990. 28, 40

[Vaa01] F. Vaandrager, *Analysis of a Biphase Mark Protocol with Uppaal*, Tech. Report CSI-R01XX, Computing Science Institute Nijmengen, 2001, to appear. 79

[Vae95] R.M.J. Vaessens, *Generalized Job Shop Scheduling: Complexity and Local Search*, Ph.D. thesis, Eindhoven University of Technology, 1995. 21

[Var98] P. Varaiya, *Reach set computation using optimal control*, 1998, EECS, UC Berkeley, Available at http://www.path.berkeley.edu/~varaiya/. 18, 109, 110

[Vel00] W. Veldman, *An intuitionistic proof of Kruskal's Theorem*, Tech. Report 0017, Department of Mathematics, University of Nijmegen, 2000. 65

[Yov97] S. Yovine., *Kronos: A verification tool for real-time systems*, Springer International Journal of Software Tools for Technology Transfer **1** (1997), no. 1/2,. 18, 136

Samenvatting

Et is better öm met 'n laaien te werken,
Äs met 'n dummen.
Nedersaksische gezegde

Om een gevaarte met ruim 290 ton vloeibaar ruw ijzer van de hoogovens naar de gieterij te vervoeren is op zich al een hele taak. Temeer als niet één maar meerdere charges tegelijk verwerkt moeten worden, welke op weg naar de gieterij een aantal behandelingen moeten ondergaan die van charge tot charge en afhankelijk van de beoogde kwaliteit staal kunnen verschillen. Het wordt er niet makkelijker op als de machines die het ijzer behandelen hooguit één charge tegelijk kunnen verwerken, en als de twee kranen die men voor elk transport moet gebruiken hetzelfde spoor moeten delen; de ene kraan kan de andere de weg versperren, charges kunnen elkaar hinderen. En bovendien mag het ruwe ijzer eer het bij de gieterij arriveert niet te ver afkoelen, en moet de toevoer zodanig zijn, dat de gieterij een continue stroom staal naar de warmwalserij kan garanderen.

Dit planningsprobleem was een van de zes casestudy's van the EU onderzoeksproject *Verificatie van Hybride Systemen*. Naast de groep Informatica voor Technische Toepassingen van de Katholieke Universiteit Nijmegen namen een tiental onderzoeksinstellingen en bedrijven uit Nederland, België, Duitsland, Frankrijk, Zwitserland, Denemarken, Zweden en Israël deel aan dit project, dat het onderzoek op het gebied van Formele Methodes moest stimuleren. De bedoeling was om inzicht te krijgen hoe met name verificatietechnieken voor hybride systemen kunnen bijdragen aan de oplossing van enkele uitdagende problemen. Het boven beschreven probleem werd voorgesteld door Sidmar, een vlakstaal-producent te Gent, België.

Een systeem is *hybride* als het gedrag van het systeem essentieel bepaald wordt door de interactie van discrete componenten zoals microcontrollers met continue processen waarin fysische grootheden en tijd een rol spelen. Met *Formele Methoden* worden in de informatica methoden en technieken bedoeld om de correctheid van een systeemontwerp mathematisch aan te tonen. Men gebruikt op wiskunde gebaseerde talen om een hard- of softwaresysteem te beschrijven, die ons dan in

staat stellen om te bewijzen of een ontwerp aan gewenste eigenschappen voldoet. Dit proces wordt *verificatie* genoemd. Typerend is te laten zien dat een systeem nooit in een onveilige toestand kan verkeren, waarbij onveilig bijvoorbeeld op een rekenfout, op fout ontvangen informatie of op het overschrijden van de maximaal toelaatbare fysische grootheid kan slaan. Een voorbeeld waarbij een afrondfout tot een verkeerd gemeten snelheid, en uiteindelijk tot een explosie heeft geleid is het mislukken van de eerste vlucht van de Ariane 5 raket.

Het met de hand bewijzen van de correctheid van een ontwerp kan zelfs voor een klein model een nogal tijdrovende en vooral foutgevoelige bezigheid zijn. *Model checking* is een van de meest gebruikte computerondersteunde technieken voor verificatie; men begint bij de beginsituatie van het systeem, en loopt dan automatisch alle mogelijke paden die het systeem kan nemen af. Zodoende wordt bijvoorbeeld geverifieerd dat het systeem nooit in een onveilige toestand kan belanden. Als traditionele algoritmen voor model checking een onveilige toestand en dus een fout vinden, kunnen deze ook informatie verstrekken hoe deze fout bereikt kan worden. Deze informatie is belangrijk om na te trekken waar de fout in het ontwerp zit. Voor bepaalde klassen modellen is gegarandeerd dat het model-checkingalgoritme altijd stopt. Men zegt dan dat het model-checkingprobleem voor deze klasse *beslisbaar* is.

Met een *automaat* wordt een formeel model bedoeld dat het gedrag van een hard- en softwaresysteem beschrijft als overgangen van één toestand naar de andere. Getimede automaten zijn een variant die toestaat om het continue verstrijken van de tijd te modelleren. Ondanks dat continue tijd tot een oneindig en zelf overaftelbaar aantal toestanden leidt, is het model-checkingprobleem voor getimede automaten beslisbaar als men met verzamelingen toestanden werkt. Model-checkingtools voor getimede automaten, zoals UPPAAL en KRONOS, hebben in de afgelopen jaren aangetoond dat zij een belangrijke bijdrage kunnen leveren aan het correcte ontwerp van soft- en hardwaresystemen.

Om terug te keren tot de staalfabriek van Sidmar. Hier gaat het er niet om om een fout in het ontwerp te vinden, maar om een oplossing voor een planningsprobleem te bepalen. We zijn niet geïnteresseerd om uit te sluiten dat ooit een onveilige toestand bereikt kan worden, maar we willen aantonen dat een gewenste toestand bereikbaar is, namelijk een toestand waarin alle charges in de gewenste volgorde en op tijd bij de gieterij gearriveerd zijn. Het is mogelijk het planningsprobleem als getimede automaat te modelleren en om een model-checkingalgoritme naar een gewenste toestand te laten zoeken. Het “tegenvoorbeeld” dat aantoont hoe deze toestand te bereiken is, kan vervolgens als oplossing van het planningsprobleem worden beschouwd.

Er zijn echter ook verschillen tussen traditionele algoritmes die optimisatieproblemen oplossen en het model-checkingalgoritme. Ten eerste maakt het model-

checkingalgoritme geen verschil tussen goede en slechte oplossingen. De kosten die aan een oplossing worden verbonden kunnen bijvoorbeeld afhangen van de totale tijd die nodig is om het bijbehorende rooster uit te voeren. Model checking wordt gebruikt om fouten te vinden, en zal stoppen zodra het de eerste fout tegenkomt. Als we hetzelfde algoritme gebruiken om planningsproblemen op te lossen, zal het stoppen zodra het een oplossing vindt, ongeacht de kosten die eraan verbonden zijn. Er is geen garantie dat dit de optimale of zelfs maar een goede oplossing is.

Een ander verschil is dat men bij het model checken de optimistische aanname maakt, dat men geen ongewenste toestand tegen zal komen, en dat men dus alle toestanden moet gaan doorzoeken. Het algoritme is derhalve zo opgezet dat het de gehele toestandsruimte efficiënt exploreert. Als we een optimisatieprobleem oplossen is daarentegen bekend dat de toestandsruimte een gewenste toestand bevat. Vaak heeft men enig idee waar een (goede) oplossing te vinden valt, en als het mogelijk zou zijn om gericht naar een oplossing te zoeken, zou dit de tijd die het algoritme nodig heeft drastisch kunnen verkorten.

Dit proefschrift onderzoekt hoe het model-checkingalgoritme voor getimede automaten uitgebreid kan worden, opdat het geschikt kan worden gemaakt voor het oplossen van planningsproblemen. Het proefschrift laat eerst zien hoe planningsproblemen als getimede automaat gemodelleerd kunnen worden. Vervolgens wordt een notie van kosten geïntroduceerd. We introduceren een gemodificeerd model-checkingalgoritme, en tonen aan dat dit desondanks gegarandeerd ooit zal stoppen, en dat het, mits een oplossing bestaat, de optimale oplossing vindt. Verder wordt aandacht besteed aan efficiënte datastructuren, en aan hoe het algoritme aangepast kan worden, zodat gericht zoeken naar goede oplossingen mogelijk wordt. Deze aanpassingen kunnen echter ook een positief effect hebben op model checking ter verificatie. Het proefschrift introduceert een model-checkingalgoritme voor een algemenere klasse hybride systemen, en onderzoekt of gericht zoeken ook voor deze klasse een positieve invloed op de performance kan hebben.

Dit proefschrift is als volgt opgebouwd. Na de inleiding in Hoofdstuk 1 wordt in Hoofdstuk 2 uitvoerig ingegaan op het getimede automaten model voor de Sidmar casestudy, en hoe met het tool UPPAAL een oplossing gevonden kan worden. Verder beschrijft het hoofdstuk hoe Job-Shop problemen – een algemene klasse planningsproblemen – vertaald kunnen worden naar getimede-automaten-modellen. In Hoofdstuk 3 worden linear geprijsde getimede automaten (LPTA) geïntroduceerd, een uitbreiding van getimede automaten waarbij de kosten stuksgewijs lineair kunnen stijgen met het verstrijken van tijd. Dit hoofdstuk presenteert een algoritme dat gegarandeerd het optimale pad naar een gegeven eindtoestand vindt, en het wordt aangetoond dat dit algoritme voor alle LPTA-modellen en elk

willekeurige eindtoestand termineert.

Het algoritme dat in Hoofdstuk 3 wordt geïntroduceerd is niet bijzonder efficient. Hoofdstuk 4 beperkt zich daarom tot de klasse van de uniform geprijsde getimede automaten (UPTA). De kosten in een UPTA stijgen uniform met verstrijken van tijd. Deze klasse bevat ondermeer problemen waar de totale tijd die de uitvoering van een rooster nodig heeft geminimaliseerd moet worden. Het hoofdstuk presenteert een algoritme dat in feite even efficient is als het oorspronkelijke model-checkingalgoritme voor getimede automaten. Verder presenteert het hoofdstuk aanpassingen van het algoritme die het toestaan om de volgorde waarin toestanden geëxploreerd worden te beïnvloeden.

Het volgende hoofdstuk 5 introduceert een efficiënte datastructuur voor de gehele klasse van LPTAs. Enkele voorbeelden tonen aan dat de datastructuren en het toestaan van gericht zoeken tot een competitief algoritme leiden. Naast voorbeelden ter illustratie worden resultaten voor andere casestudy's en benchmarks uit de literatuur gepresenteerd. Het blijkt dat gericht zoeken ook een positieve invloed kan hebben op model checken als verificatietechniek, dus ook in gevallen waar het er slechts om gaat om een fout te vinden.

Hoofdstuk 6 staat in zekere zin los van de eerdere hoofdstukken omdat het een algemenere klasse van hybride systemen beschouwt. Dit zijn systemen waar niet alleen tijd maar ook de evolutie van continue grootheden zoals temperatuur, snelheid of positie een essentiële rol spelen. In tegenstelling tot getimede automaten is het model-checkingprobleem voor deze klasse in het algemeen niet beslisbaar. Maar zelf als terminatie van het algoritme niet gegarandeerd is kan model checking een belangrijke rol bij de analyse van hydride systemen spelen. Hoofdstuk 6 beschrijft hoe men de bereikbare toestanden van een beperkte klasse systemen door veelhoeken kan overapproximeren. De overapproximatie kan dan worden gebruikt in een model-checkingalgoritme, welk op twee casestudy's toegepast wordt. Omdat in eerdere hoofdstukken bleek dat gericht zoeken een positief effect op het model checken van getimede automaten kan hebben, wordt in de tweede helft van dit hoofdstuk onderzocht of dit ook voor deze klasse het geval is. De resultaten blijken positief te zijn, maar zijn helaas verre van eenduidig.

In het laatste hoofdstuk met de conclusies wordt een kort overzicht gegeven van hetgeen dat bereikt is. Verder wordt uitvoerig stilgestaan bij de problemen die experimenteel onderzoek op het gebied van formele methoden met zich mede kan brengen. Dit hoofdstuk eindigt met een beschouwing van verder onderzoek dat uit dit proefschrift voort kan vloeien of aan dit proefschrift gerelateerd is.

Curriculum Vitae

Ansgar Fehnker was born on July 27th, 1971, in Teglingen, Lower Saxony, Germany. He started studying Mathematics at the Philips-Universität Marburg, Germany, in 1990. From 1992 on he studied at the Rijksuniversiteit Groningen, the Netherlands, and graduated in 1996 on the subject of parameter estimation of a biochemical system. He became a PhD student in 1997 at the Computing Science Institute, Katholieke Universiteit Nijmegen, under supervision of Frits Vaandrager. The author currently works as a Post Doc at the Department of Electrical and Computer Engineering of the Carnegie Mellon University in Pittsburgh, Pennsylvania.

Titles in the IPA Dissertation Series

J.O. Blanco. *The State Operator in Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1996-1

A.M. Geerling. *Transformational Development of Data-Parallel Algorithms*. Faculty of Mathematics and Computer Science, KUN. 1996-2

P.M. Achten. *Interactive Functional Programs: Models, Methods, and Implementation*. Faculty of Mathematics and Computer Science, KUN. 1996-3

M.G.A. Verhoeven. *Parallel Local Search*. Faculty of Mathematics and Computing Science, TUE. 1996-4

M.H.G.K. Kesseler. *The Implementation of Functional Languages on Parallel Machines with Distrib. Memory*. Faculty of Mathematics and Computer Science, KUN. 1996-5

D. Alstein. *Distributed Algorithms for Hard Real-Time Systems*. Faculty of Mathematics and Computing Science, TUE. 1996-6

J.H. Hoepman. *Communication, Synchronization, and Fault-Tolerance*. Faculty of Mathematics and Computer Science, UvA. 1996-7

H. Doornbos. *Reductivity Arguments and Program Construction*. Faculty of Mathematics and Computing Science, TUE. 1996-8

D. Turi. *Functorial Operational Semantics and its Denotational Dual*. Faculty of Mathematics and Computer Science, VUA. 1996-9

A.M.G. Peeters. *Single-Rail Handshake Circuits*. Faculty of Mathematics and Computing Science, TUE. 1996-10

N.W.A. Arends. *A Systems Engineering Specification Formalism*. Faculty of Mechanical Engineering, TUE. 1996-11

P. Severi de Santiago. *Normalisation in Lambda Calculus and its Relation to Type Inference*. Faculty of Mathematics and Computing Science, TUE. 1996-12

D.R. Dams. *Abstract Interpretation and Partition Refinement for Model Checking*. Faculty of Mathematics and Computing Science, TUE. 1996-13

M.M. Bonsangue. *Topological Dualities in Semantics*. Faculty of Mathematics and Computer Science, VUA. 1996-14

B.L.E. de Fluiter. *Algorithms for Graphs of Small Treewidth*. Faculty of Mathematics and Computer Science, UU. 1997-01

W.T.M. Kars. *Process-algebraic Transformations in Context*. Faculty of Computer Science, UT. 1997-02

P.F. Hoogendijk. *A Generic Theory of Data Types*. Faculty of Mathematics and Computing Science, TUE. 1997-03

T.D.L. Laan. *The Evolution of Type Theory in Logic and Mathematics*. Faculty of Mathematics and Computing Science, TUE. 1997-04

C.J. Bloo. *Preservation of Termination for Explicit Substitution*. Faculty of Mathematics and Computing Science, TUE. 1997-05

J.J. Vereijken. *Discrete-Time Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1997-06

F.A.M. van den Beuken. *A Functional Approach to Syntax and Typing*. Faculty of Mathematics and Informatics, KUN. 1997-07

A.W. Heerink. *Ins and Outs in Refusal Testing*. Faculty of Computer Science, UT. 1998-01

G. Naumoski and W. Alberts. *A Discrete-Event Simulator for Systems Engineering*. Faculty of Mechanical Engineering, TUE. 1998-02

J. Verriet. *Scheduling with Communication for Multiprocessor Computation*. Faculty of Mathematics and Computer Science, UU. 1998-03

J.S.H. van Gageldonk. *An Asynchronous Low-Power 80C51 Microcontroller*. Faculty of Mathematics and Computing Science, TUE. 1998-04

A.A. Basten. *In Terms of Nets: System Design with Petri Nets and Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1998-05

E. Voermans. *Inductive Datatypes with Laws and Subtyping – A Relational Model*. Faculty of Mathematics and Computing Science, TUE. 1999-01

H. ter Doest. *Towards Probabilistic Unification-based Parsing*. Faculty of Computer Science, UT. 1999-02

J.P.L. Segers. *Algorithms for the Simulation of Surface Processes*. Faculty of Mathematics and Computing Science, TUE. 1999-03

C.H.M. van Kemenade. *Recombinative Evolutionary Search*. Faculty of Mathematics and Natural Sciences, Univ. Leiden. 1999-04

E.I. Barakova. *Learning Reliability: a Study on Indecisiveness in Sample Selection*. Faculty of Mathematics and Natural Sciences, RUG. 1999-05

M.P. Bodlaender. *Schedulere Optimization in Real-Time Distributed Databases*. Faculty of Mathematics and Computing Science, TUE. 1999-06

M.A. Reniers. *Message Sequence Chart: Syntax and Semantics*. Faculty of Mathematics and Computing Science, TUE. 1999-07

J.P. Warners. *Nonlinear approaches to satisfiability problems*. Faculty of Mathematics and Computing Science, TUE. 1999-08

J.M.T. Romijn. *Analysing Industrial Protocols with Formal Methods*. Faculty of Computer Science, UT. 1999-09

P.R. D'Argenio. *Algebras and Automata for Timed and Stochastic Systems*. Faculty of Computer Science, UT. 1999-10

G. Fábián. *A Language and Simulator for Hybrid Systems*. Faculty of Mechanical Engineering, TUE. 1999-11

J. Zwanenburg. *Object-Oriented Concepts and Proof Rules*. Faculty of Mathematics and Computing Science, TUE. 1999-12

R.S. Venema. *Aspects of an Integrated Neural Prediction System*. Faculty of Mathematics and Natural Sciences, RUG. 1999-13

J. Saraiva. *A Purely Functional Implementation of Attribute Grammars*. Faculty of Mathematics and Computer Science, UU. 1999-14

R. Schiefer. *Viper, A Visualisation Tool for Parallel Progam Construction*. Faculty of Mathematics and Computing Science, TUE. 1999-15

K.M.M. de Leeuw. *Cryptology and Statecraft in the Dutch Republic*. Faculty of Mathematics and Computer Science, UvA. 2000-01

T.E.J. Vos. *UNITY in Diversity. A stratified approach to the verification of distributed algorithms*. Faculty of Mathematics and Computer Science, UU. 2000-02

W. Mallon. *Theories and Tools for the Design of Delay-Insensitive Communicating Processes*. Faculty of Mathematics and Natural Sciences, RUG. 2000-03

W.O.D. Griffioen. *Studies in Computer Aided Verification of Protocols*. Faculty of Science, KUN. 2000-04

P.H.F.M. Verhoeven. *The Design of the MathSpad Editor*. Faculty of Mathematics and Computing Science, TUE. 2000-05

J. Fey. *Design of a Fruit Juice Blending and Packaging Plant*. Faculty of Mechanical Engineering, TUE. 2000-06

M. Franssen. *Cocktail: A Tool for Deriving Correct Programs*. Faculty of Mathematics and Computing Science, TUE. 2000-07

P.A. Olivier. *A Framework for Debugging Heterogeneous Applications*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2000-08

E. Saaman. *Another Formal Specification Language*. Faculty of Mathematics and Natural Sciences, RUG. 2000-10

M. Jelasity. *The Shape of Evolutionary Search Discovering and Representing Search Space*

Structure. Faculty of Mathematics and Natural Sciences, UL. 2001-01

R. Ahn. *Agents, Objects and Events a computational approach to knowledge, observation and communication*. Faculty of Mathematics and Computing Science, TU/e. 2001-02

M. Huisman. *Reasoning about Java programs in higher order logic using PVS and Isabelle*. Faculty of Science, KUN. 2001-03

I.M.M.J. Reymen. *Improving Design Processes through Structured Reflection*. Faculty of Mathematics and Computing Science, TU/e. 2001-04

S.C.C. Blom. *Term Graph Rewriting: syntax and semantics*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2001-05

R. van Liere. *Studies in Interactive Visualization*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2001-06

A.G. Engels. *Languages for Analysis and Testing of Event Sequences*. Faculty of Mathematics and Computing Science, TU/e. 2001-07

J. Hage. *Structural Aspects of Switching Classes*. Faculty of Mathematics and Natural Sciences, UL. 2001-08

M.H. Lamers. *Neural Networks for Analysis of Data in Environmental Epidemiology: A Case-study into Acute Effects of Air Pollution Episodes*. Faculty of Mathematics and Natural Sciences, UL. 2001-09

T.C. Ruys. *Towards Effective Model Checking*. Faculty of Computer Science, UT. 2001-10

D. Chkliaev. *Mechanical verification of concurrency control and recovery protocols*. Faculty of Mathematics and Computing Science, TU/e. 2001-11

M.D. Oostdijk. *Generation and presentation of formal mathematical documents*. Faculty of Mathematics and Computing Science, TU/e. 2001-12

A.T. Hofkamp. *Reactive machine control: A simulation approach using χ*. Faculty of Mechanical Engineering, TU/e. 2001-13

D. Bošnački. *Enhancing state space reduction techniques for model checking*. Faculty of Mathematics and Computing Science, TU/e. 2001-14

M.C. van Wezel. *Neural Networks for Intelligent Data Analysis: theoretical and experimental aspects*. Faculty of Mathematics and Natural Sciences, UL. 2002-01

V. Bos and J.J.T. Kleijn. *Formal Specification and Analysis of Industrial Systems*. Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2002-02

T. Kuipers. *Techniques for Understanding Legacy Software Systems*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2002-03

S.P. Luttik. *Choice Quantification in Process Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-04

R.J. Willemen. *School Timetable Construction: Algorithms and Complexity*. Faculty of Mathematics and Computer Science, TU/e. 2002-05

M.I.A. Stoelinga. *Alea Jacta Est: Verification of Probabilistic, Real-time and Parametric Systems*. Faculty of Science, Mathematics and Computer Science, KUN. 2002-06

N. van Vugt. *Models of Molecular Computing*. Faculty of Mathematics and Natural Sciences, UL. 2002-07

A. Fehnker. *Citius, Vilius, Melius: Guiding and Cost-Optimality in Model Checking of Timed and Hybrid Systems*. Faculty of Science, Mathematics and Computer Science, KUN. 2002-08

Stellingen

behorende bij het proefschrift

Citius,Vilius, Melius.

van

Ansgar Fehnker

Stelling 1 *The cost-optimal reachability problem for linearly priced timed automata is decidable.* (Theorem 3.21)

Stelling 2 *The computing the cheapest path to a given location for uniformly priced timed automata is essentially as efficient as model checking of timed automata. It is justifiable to consider a UPTA as a timed automaton with a special clock.* (Section 4.3)

Stelling 3 *Results on the shortest path in a discrete graph do not need to extend to transition graphs of timed automata.* (Subsection 4.4.1)

Stelling 4 *The search order of a symbolic model checker has an influence on the number of explored symbolic states.* (Chapter 4 to 6)

Stelling 5 *A hybrid system might not get farthest with a constant extreme input* (Table 6.13). *The same holds for driving (car, bike, roller blades).*

Stelling 6 *Being a native speaker does not mean that you know your native language.*

Stelling 7 *Evolutie bekommert zich niet om keuzes van het individu.*

Stelling 8 *Elke forensende promovendus heeft tenminste één stelling over het openbaar vervoer.*

Stelling 9 *Een constitutionele monarchie is slechts in theorie een republiek met een erfelijk staatshoofd.*

Stelling 10 *The Dutch language has at least 20 different expressions for "drinking coffee".*