Semantics and Verification in Process Algebras with Data and Timing

Tim A.C. Willemse

IPA Dissertation Series 2003-05
Printed by University Press Facilities, Eindhoven

Cover design by Biep en Lú vormgeving, (`http://www.biepenlu.nl/`)

```
Semantics and Verification in Process Algebras with Data and Timing /
by Timothy A.C. Willemse. - Eindhoven :
Technische Universiteit Eindhoven, 2003.
Proefschrift. - ISBN 90-386-0672-9
NUR 993
Subject headings : process algebra / semantics / model checking /
                   logic / software verification /
                   programming languages ; muCRL /
                   programming languages ; real time
CR Subject Classification (1998) : D.1.3, D.2.4, F.3.1, F.3.2, F.4.1
```

The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics) at the Eindhoven University of Technology, supported by the Dutch Organization for Scientific Research (NWO) under contract 612-014-008.

Semantics and Verification in Process Algebras with Data and Timing

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Technische Universiteit Eindhoven,
op gezag van de Rector Magnificus, prof.dr. R.A. van Santen,
voor een commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen op
donderdag 20 februari 2003 om 16.00 uur

door

Timothy Ariën Carol Willemse

geboren te Grubbenvorst

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr. J.C.M. Baeten
en
prof.dr.ir. J.F. Groote

Acknowledgements

I am indebted to my promotors Jos Baeten and Jan Friso Groote, my daily supervisor Kees Middelburg and Rance Cleaveland for taking part in the manuscript committee. I am very thankful for their criticism, and the comments they had on this thesis have led to many improvements. On this occasion, I also like to thank Wan Fokkink, Frits Vaandrager, Sjouke Mauw and Martin Rem for agreeing to take part in the defence committee. Jos Baeten is thanked for acting suprised last summer when confronted with the first full version of this thesis. I also thank him for his careful guidance during these years and his advice on many research questions. Jan Friso is thanked for introducing me to μCRL, the area of symbolic model checking and his enthusiasm in general. He is also a co-author of the paper that led to Chapter 3. I am grateful to Kees Middelburg for his professional guidance throughout these years and his ability to make my thoughts intelligible for others.

During these past four years, the very pleasant and positive atmosphere at the Formal Methods group in Eindhoven, headed by Jos Baeten, has helped me to skip various illnesses a student is supposed to experience, such as the infamous *Ph.D. depression*. Many thanks go to my roommates, for putting up with my silly musical escapades, art impressions and (other) work-related issues. The first and largest part of these past four years, I shared my office with Suzana Andova and Martijn Oostdijk. Suzana's happy mood and the scribblings I shared with her on our Wyte Board have been of great importance to me. As a co-founder of TafkaTaM, Martijn is thanked very much for his inspiration and influence. Also, his ability to turn all my research problems into computer programs has left me with a deep sense of awe. After Suzana and Martijn left, I shared my office with "go-buddy" Georgi Jojgov and Ka Lok Man. Their continuing support during the more stressful parts of these past four years is very much appreciated. Marc Voorhoeve is thanked for occasionally wandering into my office and sparking tiny new revolutions in computer science. Ana, Anne-Meta, Cas, Desiree, Dmitri, Dragan, Erik, Erik, Elize, Emmy, Francien, Gia, Harm, Jerry, Jos, Kees, Louis, Michael, Nicu, Rob, Ruurd, Sjouke, Tijn, Victor are thanked for making these past four years very much fun. The social activities, such as borrels, Sinterjos, Kersttapas, biking trips and playing games of Go are memorable indeed. The research school IPA, and especially the IPA-team, is thanked for the enjoyable social activities and its educational efforts.

The life of a Ph.D. student is not necessarily a life cluttered only with research. The Eindhoven Studium Generale working group on music, headed by David Ernst, helped me to stay in touch with music in the broadest sense. The two-weekly Ph.D. meetings (TAO) I organised together with Isabelle Reymen (and later with Susanne Loeber), being a member of the promotion-council for IPA and being a Ph.D. representative helped me to focus on and gain experience

in many organisational issues. I especially thank Sjouke Mauw for the countless occasions on which he diverted my attention to activities other than my own research project. His meddlings resulted in us organising the VFM-symposium in 1999, the Testdag in 2001, and even writing scientific output.

I thank the *Dutch garlic-boys in the frying-pan of lightning*, Hakselaar (Jeroen), Bruut (Bert), Aad (Adriaan) and Germ (Jeroen) for enabling me to berserk mountains in Europe and pitch our tents in the strangest possible places during winter. Our journeys allowed me to occasionally break with daily life and learn to appreciate nature, but also civilisation and luxury, so much more whenever I return home. Many thanks go to the RISKy-whiskey group for teaching me to be humble and accepting my losses. I thank my brother for unmentionably many things, not the least for being my opponent in many squash games. Lutsen, from *Biep en Lú Vormgeving* is thanked for the effort he put in designing a groovy cover for this thesis. Finally, I thank all my friends for their moral support.

Without the current roof over my head, I most likely would have finished this thesis a year earlier. However, since Marcella and I now own a pile of neatly organised stones, I feel obliged to (once again) thank all our friends that helped us to make our house a warm and cosy place to live, allowing me to finish my thesis without too much delay. Special thanks go to my mum, dad and Ine for their moral and physical support, even though they had their reservations with the bricks Marcella and I bought. Also, Victor Bos, Georgi Jojgov and Ana Sokolova are thanked once more for the (almost excessive) amount of time they put into rearranging the layout of our house, patching and sanding walls and preparing items that had never before known colour.

Finally, I would like to thank Marcella, *Mupke*, for supporting me and believing in me. Her support, especially during the five months I spent on typing this thesis while living our temporarily diminished social life, is deeply appreciated.

Eindhoven, January 2003

Contents

Chapter 1

Introduction

This thesis is concerned with the use of process algebras for verification and semantics. To place this thesis in a broader perspective, we first give a brief account of the history of computer science, thereby illuminating the topics of this thesis to the reader, unfamiliar with these topics. Section 1.2 serves as an introduction to *process languages* without going into theoretical issues and details. Finally, Section 1.3 provides a short introduction to each chapter of this thesis.

1.1 A Brief History of Computing and Computer Science

Without a doubt, the invention of the computer has had an impact on human civilisation, possibly equal in proportion to the first fire ever lit intendedly by mankind. Even though the concepts for modern-day computers have been around no longer than half a century, the path towards its development was initiated several millennia earlier.

One of the earliest devices, aiding in conducting complex computations, is the *abacus*. Records show that this instrument came into fashion as early as 3000 B.C. and is still used to this date in various cultures. Although the abacus itself only constitutes a register, and has no means for mechanising parts of computations, schemes have been devised for performing various mathematical operations, such as additions, subtractions, multiplications and divisions. Such schemes can be considered crude forms of algorithms.

It took more than four millennia for the next notable inventions to start paving the way to modern computers. Napier, best known for his invention of logarithms, devised a number of mechanical aids for doing arithmetic. Napier's rods (also known as “Napier's bones”) supported addition, multiplication and powers. Although the aid was well-known, it never became very popular. Even so, Napier's mechanical aids kick-started the development of mechanical machinery for conducting ever more complex computations. It is Schickard who is recognised for pioneering the construction of a mechanical calculator in the 17^{th} century. He combined Napier's bones with a method for adding up the partial products that resulted from the use of Napier's bones, resulting in an automated process for performing multiplications.

More than a decade later, Pascal invented his own adding machine, known as the *arithmetic machine*. The device could add up to 999999.99 and was also able to subtract. Based on the ideas of Pascal on for instance the mechanism for carrying out a carry, Leibnitz constructed his *mechanical multiplier*. This device improved on Pascal's arithmetic machine by performing multiplication, division and calculating square roots.

The start of the 19th century witnessed the development of the first memory device, invented by Joseph-Marie Jacquard, in the form of perforated cards for a programmable loom. His ideas were subsequently used by Babbage for programming his steam powered *analytical engine*. No working model of this latter device was ever produced, but from the drawings he made (covering over 100 square metres of paper), it can be deduced it was to be capable of performing basic arithmetic functions for any mathematical problem. A summary of Babbage's presentation on his ideas concerning this analytical engine was published in French by the Italian Menebrea. Ada Byron, Lady Lovelace, translated this summary and, on the suggestion of Babbage himself, added many ideas of her own. She suggested to Babbage writing a plan for how the engine might calculate Bernoulli numbers. This plan, is now regarded as the first "computer" program.

Several noteworthy inventions passed, such as Hollerith's tabulating machine and Zuse's computing machines. Hollerith founded his own company at the end of the 19th century. This company later became part of the International Business Machine Corporation, also known as *IBM*. Zuse is recognised for being the first person ever to construct an automatically controlled calculating machine. He built four different types of computing machines, i.e. the Z1, Z2, Z3 and the Z4. He later formed his own company for the construction and marketing of his designs, which, after a series of merges and takeovers finally ended up as part of the computer division of Siemens. Neither Zuse's, nor Hollerith's designs were all-electronic computers. The first computer of this kind was the ENIAC, devised in 1946.

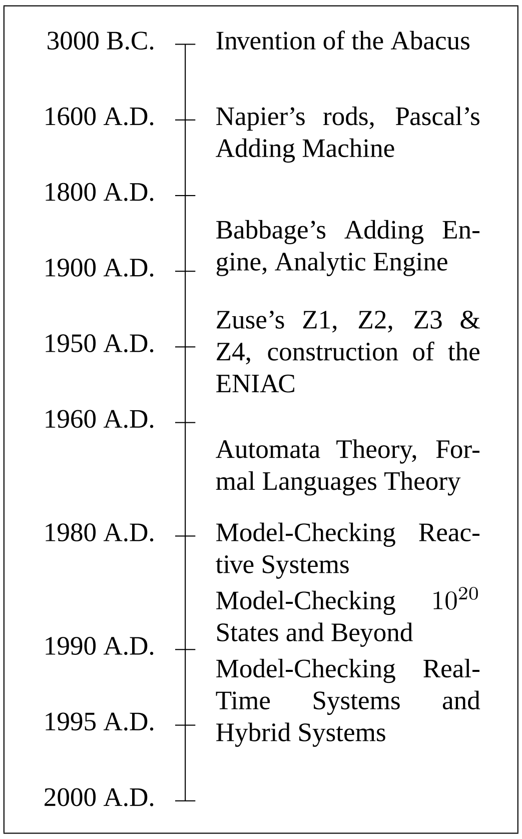

3000 B.C.
Invention of the Abacus
1600 A.D.
Napier's rods, Pascal's Adding Machine
1800 A.D.
Babbage's Adding Engine, Analytic Engine
1900 A.D.
Zuse's Z1, Z2, Z3 & Z4, construction of the ENIAC
1950 A.D.
1960 A.D.
Automata Theory, Formal Languages Theory
1980 A.D.
Model-Checking Reactive Systems
Model-Checking 10^20 States and Beyond
1990 A.D.
Model-Checking Real-Time Systems and Hybrid Systems
1995 A.D.
2000 A.D.

Figure 1.1: Milestones in history

It was the publication of John von Neumann's paper "Theory and Techniques of Electronic Digital Computers" that finally sparked the revolution in the development of the modern-day digital computer.

However, it was not until 1962 that computer science came into its own as a discipline. From then on, the science of computing rapidly made progress. The '60s saw the advent of automata theory and the theory of formal languages and, in general, a mathematical basis for the analysis of algorithms started to get some attention. In other words, a theoretical basis for computer science was initiated. The '70s saw the introduction of super computers, and major advances in algorithms and computational complexity, firmly establishing a foundation for computer science. By then, computers were used both for performing large calculations and for automating simple tasks. Both applications required a different type of software. Programs designed for calculations could, on an abstract level, often be specified using pre-and post conditions. Such programs (known as *transformational* programs) basically converted a given input in such a way that on termination of the computation, the output contained the desired information. Opposed to this type of programs, *reactive systems*, used in automation, were meant to run indefinitely, constantly responding to stimuli they received from their environment. It is the latter class of programs this thesis is concerned with.

Reactive systems are omnipresent in nowadays society. The average person quickly has over a dozen of devices that can be classified as such. Notable examples are mobile phones, televisions, stereo equipment, micro-wave ovens, etc. However, reactive systems are also abundantly present in hospital-equipment, aeroplanes, chemical and nuclear power plants, missiles, etc. These latter examples are so-called *safety-critical* systems. Safety-critical systems must meet the strictest requirements, guaranteeing the safe functioning of the system under all circumstances. The literature documents a number of mortal incidents, as a result of the faulty programming of safety-critical systems. A well-known example is the Therac-25 radiation machine.

Over the past decades, the awareness of our dependency on reactive and safety-critical systems has stimulated and directed research towards the development of methods and techniques, enabling us to make more than a mere educated guess on a system's properties. In the '80s and '90s, notable techniques, such as testing, theorem proving and model-checking have emerged, allowing for the manual, automated or even automatic verification of properties of systems. Already, these techniques have been applied successfully in stages of the development of many kinds of reactive systems. However, with the ever increasing complexity of these systems, new techniques must be developed for dealing with this increase in complexity. This complexity increase is not only the result of the increasing number of tasks that must be automated, but is also caused by additional requirements, such as timeliness and data-dependency of the system. The data-dependencies can easily cause an explosion in the number of states a system can be in. This is readily illustrated by the "process" of ten persons counting, independently, up to a hundred: the total number of "states" we can distinguish is already astronomical. The class of reactive systems, studied in this thesis are time and data-dependent systems.

Systems with quantitative timing-requirements are generally known as *real-time systems*. The trend of manufacturing ever faster processors has opened up possibilities previously unimaginable. Current processors can take a decision in a split-second, making them suitable for time-critical application areas, such as avionics, state-of-the-art communications protocols, etc. These processors are increasingly used in situations where decisions have to be made on the basis of continuously changing variables. For example, a controller for a bottle-filling system must decide instantly to stop filling a bottle when it is full, and start transporting a new bottle to the filling system. Such real-time systems, i.e. systems with one or more dependencies on continuous variables, are generally considered part of a class named *hybrid systems*. This thesis is concerned with both real-time systems and hybrid systems.

1.2 Process Languages

The previous section already hinted at our great dependency on computerised systems. Their correct and safe functioning is therefore a matter of utmost importance. As we already mentioned, techniques have been developed for verifying important aspects of systems. Notwithstanding the beauty of mathematics, used for describing the natural sciences, a different kind of mathematics is needed for describing reactive systems. We refer to these languages as *process languages*.

Unfortunately, today's situation much resembles a post-tower-of-Babel situation: many different process languages have been proposed, but no single language ever surfaced as *the* process language. Roughly, we can distinguish two types of languages: visually oriented and text oriented. Visually oriented languages (e.g. SDL [71, 69] and MSC [70, 96]) allow for a non-linear, spatial presentation of the important aspects of a system. They use graphical notations, often consisting of boxes, arrows, circles and other meaningful objects, representing valuable information. Text-oriented languages (see e.g. the language Promela [67], ACP [18] and most logics), on the other hand, are much more linear, using words and symbols to represent system information. For small systems, visual languages may hold the whip hand when it comes to intuition; however, the larger a system gets, the harder it is to give an intuitive and concise description of the important aspects of a system, regardless of the type of language. In this thesis, we mainly focus on a text-oriented language.

Most process languages are designed to focus on specific aspects of a system. In this thesis, we are most concerned with the *dynamic* aspects of a system, rather than the *static* aspects of a system. This means, we are primarily interested in the behaviour exhibited by a system, i.e. we can identify a notion of *state* and a notion of *evolution*. The latter is often represented as a *transition* between states. Different process languages deal differently with these concepts, but we can roughly distinguish two approaches: one approach focuses only on *runs* or *traces* of a system, whereas the other approach also considers the moments of choice of a system. Needless to say, using the second approach, a much finer image of a system's behaviour can be obtained. This does not necessarily mean this is always advantageous, but most of the information obtained using the former approach can also be obtained using the latter approach, yet not vice-versa. All three process languages considered in this thesis, being μCRL_t [49, 108], Timed Automata [7, 8] and Hybrid Automata [62], are interpreted according to the second approach.

The language μCRL_t is a text-oriented process language. Terms, expressible in this language, can be related to other terms in the language via calculations using axiomatic laws. For example, for a given system, its envisioned behaviour can be (should be) related to a real-life implementation of the system using the axiomatic laws. Process languages, allowing for such form of reasoning are referred to as *process algebras*. The use of process algebras is not restricted to such applications; often they are used to study the basics of a class of systems by studying the axiom system and various interesting derived properties. The languages of Timed Automata and Hybrid Automata are more visual-oriented, but are also equipped with a textual syntax. These automata languages are considered easy to understand, and have been successfully used on various occasions. All three languages allow for the specification of real-time and time-dependent systems. The relation between μCRL_t and both types of automata languages is investigated in this thesis.

1.3 Overview of this Thesis

The previous two sections marked several issues as points of attention of this thesis. In this section, we give an outline of this thesis and provide a brief account of the content of each chapter.

The common denominator of this thesis is the process language μCRL_t. This language is a real-time process algebra allowing for the specification of timed *and* untimed data-dependent processes. Since this thesis is most likely to be unreadable without some basic understanding of the syntax and semantics of μCRL$_t$, Chapter 2 aims at filling this gap for the unfamiliar reader. Given the variety of axiom systems named μCRL and μCRL$_t$, we take two sources as our starting point and repeat the syntax, semantics and some meta-theoretical results of the languages pCRL, pCRL$_t$ and μCRL$_t$. The relation between these languages is clarified in Chapter 2.

In Chapter 3, we address the issue of verification of data-dependent systems. We define an expressive logic, extending the standard modal μ-calculus [75, 29] with first-order quantifiers. This logic is then taken as our language for stating properties for verification of systems specified using the untimed fragment of μCRL$_t$. Using a formalism, known as *fixpoint equation systems*, we describe an algorithm for fully automatically verifying the satisfaction of properties of a given system. The problem we deal with in this chapter is in general undecidable, hence, termination of our algorithm is not guaranteed. However, we show several interesting examples, illustrating the applicability of this type of verification in practice. This chapter is joint work with Jan Friso Groote and is based on [59].

Chapter 4 is in some sense orthogonal to Chapter 3: rather than verifying a system's properties *a posteriori*, properties are imposed on a system *a priori*. In this chapter, we start out with the basic idea of restricting the reachable states of a system by systematically allowing or disallowing events to occur. Instead of considering only a single initial state of a system as a starting point, as is done in the majority of the existing literature on similar subjects, we take the point of view that any *allowed* state can be an initial state. As a result, such a system is easily replaced with another, perhaps slightly different system, with similar characteristics. This perspective fits in nicely with the methodology of component-oriented software design. We focus on several conditions that allow us to establish non-deadlocking restrictions of a system's behaviour more easily.

The relations between μCRL$_t$ and timed automata and hybrid automata, mentioned already at the end of the previous section, is subject of investigation in Chapters 5 and 6. We first take a closer look at timed and hybrid automata in Chapter 5. There, we develop a semantics for both languages, using a different model than is standardly employed. These semantics are in turn related to the standard semantics of these languages. The results, obtained in this chapter reflect the inherent differences and conformity between these languages and μCRL$_t$. We subsequently interpret (classes of) these languages in μCRL$_t$ and prove correctness of these interpretations. The results, established in this chapter may in the near future add to the development of additional tool support for μCRL$_t$, and provide an interesting perspective on the use of μCRL$_t$ for specifying hybrid systems. These two chapters are based on [125].

In Chapter 7, we study the applicability of μCRL$_t$. By investigating a relatively complex system (in fact a hybrid system), we assess the current state of the art for μCRL$_t$. By continuing the analysis of this system beyond the capabilities of the current theory of μCRL$_t$, we arrive at a general picture of its added value and, more importantly, its limitations. The case study appeared in a slightly different form in [123, 124]. We wrap up this chapter with some pointers for future research and directions for the further development of μCRL$_t$.

Each chapter itself is preceded by a short introduction, providing a motivation for the theory or applications it describes. The last chapter of this thesis contains an English summary and directions for future research. We end with a Dutch summary of the preceding chapters.

Chapter 2

Data, Time and Process Algebras

In this chapter, we introduce a family of process algebras, viz. pCRL [52, 50, 84], pCRL$_t$ [49, 108] and μCRL$_t$ [49, 108]. All three process algebras can be used for the specification of data-dependent processes, however, only the latter two process algebras allow for the specification of time-dependent systems. Those, familiar with the above-mentioned process algebras may find that we do not explicitly deal with μCRL [52, 53]. The reason for this is twofold: on the one hand, the process algebra μCRL$_t$ supersedes μCRL. This means that any μCRL expression is immediately also a μCRL$_t$ expression with the same intended behaviour. Therefore, a treatment of the theory of μCRL could be considered unnecessary. On the other hand, we treat the above-mentioned process algebras in a uniform way, as directed by [50, 84, 108]. Unfortunately, μCRL itself has never been (re-)formulated in this fashion.

Historically, the language μCRL (and its successor μCRL$_t$) is rooted in another language, called *Common Representation Language*, or CRL for short. CRL was designed in the European RACE project SPECS (Specification and Programming Environment for Communication Software). The purpose of this project was to design a language containing many features of (at that time) accepted specification languages. Then, by providing a sound translation of other languages to CRL, tool support based on CRL would be sufficient for analysing other languages. Unfortunately, these goals turned out to be too ambitious, and were never completely fulfilled.

The language μCRL (and thus μCRL$_t$) is a stripped down version of CRL, having far less language features. The process specification part of μCRL is closely related to the algebraic theory of ACP [18], and PSF [95].

The outline of this chapter is as follows. Section 2.1 is a concise introduction into the data-theory for pCRL, pCRL$_t$ and μCRL$_t$. Subsequently, we discuss the theory of pCRL in Section 2.2, repeat several theoretical results and illustrate the use of pCRL in several small examples. In Section 2.3, we extend the theory of pCRL with quantitative timing information and in Section 2.4, we extend the resulting theory pCRL$_t$ with parallelism. Finally, in Section 2.5, we discuss a special format for writing pCRL, pCRL$_t$ and μCRL$_t$ expressions.

2.1 Data

Data plays a decisive role in many systems. Best-known examples are communications protocols, but in general, any system capable of exchanging information with its environment uses data. The presence of data in a system has an undeniable effect on the behaviour of a system. Hence, abstracting from the data in a specification is in many situations undesirable.

As mentioned in the prelude to this chapter, the process languages pCRL, $p\text{CRL}_t$ and μCRL_t are designed to support the specification of data-dependent systems. The data, and the data-types, used in e.g. μCRL_t-specifications are specified using the theoretically sound and flexible (yet impractical) framework of *(equational) abstract data-types*. A data-type is specified as a *data-sort*, using the reserved word **sort**. To each sort, a number of constructors, (preceded by the reserved word **func**) and functions (preceded by the reserved word **map**) are associated. A sort represents a non-empty set of data elements. Whenever a sort D is declared without any constructors with target sort D, then it is assumed that D may be arbitrarily large. The equations associated to a data-sort define which data expressions are assumed equal. Preceding these equations is the keyword **rew**, and, possibly **var**, declaring the variables used in the equations. Specification techniques, similar to the one sketched here, can be found in e.g. [95, 24].

Example 2.1.1. An example of a data-type is the set of natural numbers, on which we define addition and multiplication. The equations express elementary laws for addition and multiplication

sort	*Nat*		
func	$0:$		$\rightarrow$ *Nat*
	$s:$	*Nat*	$\rightarrow$ *Nat*
map	*add* :	*Nat* $\times$ *Nat*	$\rightarrow$ *Nat*
	mult :	*Nat* $\times$ *Nat*	$\rightarrow$ *Nat*
var	$m, n :$ *Nat*		
rew	*add*$(0, n)$	$= n$	
	add$(s(n), m)$	$= s($*add*$(n, m))$	
	mult$(0, n)$	$= 0$	
	mult$(s(n), m)$	$=$ *add*$(m,$*mult*$(n, m))$	

of natural numbers. Confusingly, the equations are preceded by the keyword **rew**, suggesting some underlying rewrite strategy associated to the equations.

Given the rather involved nature of specifications for even the most elementary data-types, we refrain from using abstract data-types in this thesis. Instead, we simply assume the existence of an appropriate data signature. For the exhibition of the theory, discussed in this chapter, we assume a data signature consists of a set S of sort symbols and a set F of function declarations. We assume disjoint, infinite sets of variables V_s for the sort symbols $s \in S$. The collection of all variables is $V = \bigcup_{s \in S} V_s$. The set of terms of sort s is denoted T_s. In particular, we assume the existence of a sort symbol $\mathbb{B}$ for the booleans with function declarations $\mathsf{t}\colon \rightarrow \mathbb{B}$ and $\mathsf{f}\colon \rightarrow \mathbb{B}$, and the usual boolean connectives $\neg$, $\wedge$ and $\vee$. For each sort symbol s, we assume a data algebra with universe $\mathcal{D}_s$. The set $\mathcal{D}_{\mathbb{B}}$ has two elements, consisting of the interpretation of t and the interpretation of f.

For the languages $p\text{CRL}_t$ and μCRL_t, we furthermore assume the existence of a sort symbol $\mathbb{T}$, consisting of time elements. We require that it is totally ordered, denoted $\leq$, and contains a least element, denoted $\mathbf{0}$.

Bibliographic notes

The process algebras, used in this thesis are not unique with respect to the ability to specify data-dependent processes. Noteworthy other examples are LOTOS [24] and PSF [95].

The data language of the formal description technique LOTOS is based on the data language ACT ONE [40, 41]. Its approach is based on an equational specification of data types, just as the data language for μCRL. The semantics of ACT ONE is given in an initial algebra model, whereas the language for the data part in μCRL is based on a *model class algebra*. This means there are fundamental differences between the data language of μCRL and ACT ONE. Most notably, ACT ONE does not distinguish between constructor elements of a sort and non-constructor elements of a sort, whereas μCRL does (note that the original definition of μCRL did *not* make this distinction). The distinction between constructor elements and non-constructor elements is needed to circumvent several problems, identified in e.g. [49]. For instance, in an initial algebra model, it is not possible to declare partial functions and a non-empty sort *without* defining any constructor for this. Thus, using an initial algebra model, one cannot specify a sort *Message* of messages without describing all its messages, or specify an arbitrary large time domain (as is done in μCRL$_t$).

Remark that many tools, such as Cæsar Aldébaran [45, 46], operating on LOTOS specifications, have patched the official LOTOS semantics, allowing them to distinguish between constructors and non-constructors.

ACT ONE is slightly more extensive than the data language of μCRL. For instance, it allows for parameterised type specifications, type renaming and conditional equations. Such constructs are not present in the data language of μCRL, which basically has a very simple data language.

The semantics of the PSF data language is, like ACT ONE, based on an initial algebra model. Many of the features we already mentioned for ACT ONE are also present in the data language for PSF. Apart from these, it also offers modularity and hiding mechanisms, and entire libraries of data types have been developed. As such, the data language of PSF is much more complex than that of μCRL.

2.2 Syntax and Semantics of pCRL

The language pCRL (and its axiomatic theory) is tailored to dealing with sequential, concrete, data-dependent processes. In fact, pCRL is an extension of BPA$_\delta$ [18], adding data-quantification and conditional operators. The building blocks of pCRL consist of a constant, a set of atomic actions, and several operators combining these to construct larger terms.

Definition 2.2.1 (*Signature of pCRL*).
The signature of the theory pCRL consists of both a data signature and a process signature. We restrict our attention to the process part of pCRL. For a more detailed account of the data signature, we refer to e.g. [52, 49]. The process signature consists of the sort symbol $\mathbb{P}$ and the function declarations given below:

1. *action declarations* $\mathsf{a}{:}s_1 \times \cdots \times s_n \to \mathbb{P}$ for given sort symbols s_i ($i = 0, \ldots, n$) from the data signature,

2. *inaction* δ: $\rightarrow \mathbb{P}$. The constant δ is meant to represent situations in which progress is blocked, due to a lack of alternatives for continuation,

3. *alternative composition* $+ {:} \mathbb{P} \times \mathbb{P} \rightarrow \mathbb{P}$. The process term $p + q$ is intended as the non-deterministic choice between the process terms p and q,

4. *sequential composition* $\cdot {:} \mathbb{P} \times \mathbb{P} \rightarrow \mathbb{P}$. The process term $p \cdot q$ represents a succession of process term p by process term q,

5. *conditional operator* $\lhd\ \rhd {:} \mathbb{P} \times \mathbb{B} \times \mathbb{P} \rightarrow \mathbb{P}$. The process term $p \lhd b \rhd q$ is intended to represent process term p whenever b is equal to t, and process term q when b is equal to f.

6. *alternative quantification* $\sum_v {:} \mathbb{P} \rightarrow \mathbb{P}$. The process term $\sum_v p$, where $v \in V$, is intended to represent the (possibly infinite) choice between the (possibly infinite number of) process terms $p[d_i/v]$ for any data term d_i of the sort of v.

The common terminology is to refer to the above function declarations as *operators*. The set, containing all action declarations is written *Act*. We distinguish process terms and action terms, the latter being terms of the form $\mathsf{a}(d_1, \ldots, d_n)$, for a given action declaration $\mathsf{a}{:}s_1 \times \cdots \times s_n \rightarrow \mathbb{P}$ and data terms d_i of sort s_i ($i = 1, \ldots, n$). Process terms are composed of action terms, data terms, variables and the operators defined above. We denote the set of all action terms by AT, and write AT_δ for the set $AT \cup \{\delta\}$. The set of all process terms is written $T_\mathbb{P}$. We assume an infinite set $V_\mathbb{P}$ of process variables, disjoint from the set V of data variables. The subset of process terms possibly containing bound and free data variables, yet no process variables, is called the set of *process-closed terms*. We typically write $p, q,$ etc. for process-closed terms. For arbitrary process terms, we write $x, y,$ etc.

The binding strength of the various pCRL operators is given by Eqn. (2.1), where we use $\geq$ to denote that the lefthand-side operator binds stronger than the righthand-side operator.

$$\cdot \ \geq\ \lhd\rhd\ \geq\ \sum_v\ \geq\ + \tag{2.1}$$

A process algebra typically characterises the intended effects of its operators by means of equational laws, establishing identities between process terms that should be considered related. Apart from the intuition on the operators these laws develop, equational reasoning can be used for the rewriting of process terms to less complex process terms. We first list the axioms for pCRL, and subsequently provide several illustrative examples of applications of pCRL in practice.

Definition 2.2.2 (*The axiom system of pCRL*).
The axioms of pCRL are the axioms listed in Table 2.1.

Although we allow recursive equations of the form $X(d_1, \ldots, d_n) = p$, where X is a recursion variable, parameterised by the data values $d_1, \ldots, d_n$ and p is a process term, possibly containing the recursion variable X, recursion is not formalised in the exposition of this thesis.

The axioms of pCRL require some explanation. Our remark that pCRL builds on the theory of BPA$_\delta$ is justified by the existence of the axioms A1 through A7. From BPA$_\delta$, we know the operator $+$ is commutative and associative, and the operator $\cdot$ is associative. This makes sense. The choice to execute either a system, represented by process term p or a system, represented by

A1	$x + y$	$=$	$y + x$
A2	$x + (y + z)$	$=$	$(x + y) + z$
A3	$x + x$	$=$	x
A4	$(x + y) \cdot z$	$=$	$x \cdot z + y \cdot z$
A5	$(x \cdot y) \cdot z$	$=$	$x \cdot (y \cdot z)$
A6	$x + \delta$	$=$	x
A7	$\delta \cdot x$	$=$	δ
PE	$p \lhd \mathit{eq}(v, w) \rhd \delta$	$=$	$p[v/w] \lhd \mathit{eq}(v, w) \rhd \delta$
SUM1	$\sum_v x$	$=$	x
SUM3	$\sum_v p$	$=$	$\sum_v p + p$
SUM4	$\sum_v (p + q)$	$=$	$\sum_v p + \sum_v q$
SUM5	$(\sum_v p) \cdot x$	$=$	$\sum_v (p \cdot x)$
SUM12	$(\sum_v p) \lhd b \rhd \delta$	$=$	$\sum_v p \lhd b \rhd \delta$
C1	$x \lhd \mathsf{t} \rhd y$	$=$	x
C2	$x \lhd \mathsf{f} \rhd y$	$=$	y
C3	$x \lhd b \rhd y$	$=$	$x \lhd b \rhd \delta + y \lhd \neg b \rhd \delta$
C4	$(x \lhd b_1 \rhd y) \lhd b_2 \rhd y$	$=$	$x \lhd b_1 \wedge b_2 \rhd y$
C5	$x \lhd b_1 \rhd \delta + x \lhd b_2 \rhd \delta$	$=$	$x \lhd (b_1 \vee b_2) \rhd \delta$
C6	$(x \lhd b \rhd y) \cdot z$	$=$	$x \cdot z \lhd b \rhd y \cdot z$
C7	$(x + y) \lhd b \rhd z$	$=$	$x \lhd b \rhd z + y \lhd b \rhd z$
SCA	$(x \lhd b \rhd \delta) \cdot (y \lhd b \rhd \delta)$	$=$	$x \cdot y \lhd b \rhd \delta$

Table 2.1: Axiom system for pCRL, where process-closed $p, q \in T_{\mathbb{P}}$, $x, y, z \in V_{\mathbb{P}}$, $v, w \in V$ and $b, b_1, b_2 \in T_{\mathbb{B}}$

process term q is not determined by the order in which the process terms are combined using the $+$ operator. Hence, commutativity holds. Similar arguments immediately justify the associativity of both $\cdot$ and $+$. From here on, we omit parentheses in the context of the operators $\cdot$ and $+$ wherever possible.

Remark 2.2.3. *On many occasions, ACP and μCRL adepts follow the mathematical convention of suppressing as many symbols as possible, resulting in the notation pq when actually $p \cdot q$ is meant. In this thesis, we do not adopt this convention and favour the notation $p \cdot q$.*

Clearly, the axioms, defining the interplay of the inaction constant and sequential composition and alternative composition (i.e. axioms A6 and A7), define δ as the neutral element for alternative composition and as a left-zero for sequential composition. Thus, inaction cannot be *chosen* as an alternative, as, in the context of a viable alternative, inaction is redundant. The more common name *deadlock* for the symbol δ is therefore slightly misleading, as it may lead one to think that the process term $\delta + p$, where $p \neq \delta$ can deadlock non-deterministically. In the context of sequential composition, however, the intended behaviour of inaction is essentially indistinguishable from the intuition one has for deadlock.

The added value of pCRL over BPA$_\delta$ lies in its ability to cope with data. The solution, adopted in pCRL, is to introduce a binding operator $\sum$ and parameterised actions. The $\sum$ operator is generally referred to as the *sum-operator*. The sum-operator introduces a number of difficulties, often associated with binding operators. The unintended binding of a data variable v when substituting process-closed terms p (where $v \in FV(p)$) for process variables x is prevented by adopting the convention of α-conversion, known from e.g. the lambda calculus. Thus, before such substitutions can occur, we assume all bound variables in the process-closed term p have been replaced by fresh variables. Consequently, we consider process terms modulo α-conversion.

We briefly explain some of the axioms for the sum-operator. Axiom SUM1 concisely expresses that we can eliminate redundant sum-operators, i.e. whenever a bound variable does not occur (freely) in the process variable taken as an argument for the sum-operator, the quantifier can be eliminated. SUM3 expresses that each process-closed term, taken as an argument for the sum-operator is considered an alternative. Axiom SUM4 expresses that considering the choice between (in)finitely many alternatives boils down to considering infinitely many alternatives of choices. From the axioms SUM3, SUM4, and SUM5 we can understand that the sum-operator generalises the $+$ operator of BPA$_\delta$.

Data variables are useful, only if on the basis of their values, decisions can be made. The ternary conditional operator allows us to specify data-dependent process-terms. The language pCRL is not restricted to the use of a binary boolean system, but allows for more exotic systems. This explains the additional axioms C3 through SCA. Reading $p \lhd b \rhd \delta$ as the process term p only if b equals $\mathtt{t}$ (i.e. if b then p), the axioms for the conditional operator are self-explanatory.

Remark 2.2.4. *In case the conditional operator has the form $p \lhd b \rhd \delta$, we use the convention to write $[b] ::\!\to p$. The boolean b can be considered a guard for process p.*

Finally, from axiom PE, we can deduce a process term depends on the value of data variables, rather than on the naming of these variables. The data operator *eq*(v, w) for variables v and w is the built-in equality for the sort of variables v and w (see also [84, 127]). Note that in [50], it is shown that this axiom is needed to obtain relative completeness of pCRL, but apart from this, can be considered a technicality of the language pCRL.

Remark 2.2.5. *As we already mentioned in the introduction to this chapter, we here follow [50, 108] in our exposition of pCRL. Several other axiom systems for pCRL, μCRL and μCRL$_t$ have been defined, see e.g. [52, 84, 49, 53]. The most apparent differences are in the treatment of the sum-operator and the conditionals. The approach, outlined in this thesis, allows for more exotic forms of logics to be used in conditionals and considers process terms modulo α-conversion. Many of the other approaches allow only a two-valued logic and define axioms to cope with situations that are easily solved with α-conversion.*

The underlying logic used in addition to the axioms is an extension of the well-known equational logic, and is referred to as *generalised equational logic* (see Table 2.2). This logic is first discussed in [50], and extends on the standard equational logic by defining appropriate rules for substitution such that bound variables are never substituted for and free variables do not become bound by the sum-operator. We illustrate the syntax of pCRL using several small examples.

Example 2.2.6. Below, we illustrate the use of sequential and alternative composition in writing specifications. We model a part of a gameshow in which a candidate is asked to pick one of three

$$t = t' \qquad \text{for every } t = t' \in E$$

$$\frac{t = t'}{t[e/v] = t'[e/v]} \qquad \text{for every } v \in V_s \text{ and } e \in T_s$$

$$\frac{t_1 = t_1' \cdots t_n = t_n'}{F(t_1, \ldots, t_n) = F(t_1', \ldots, t_n')} \qquad \text{for every } F{:}s_1 \times \cdots \times s_n \to s' \in \Sigma$$

$$t = t \qquad \frac{t = t'}{t' = t} \qquad \frac{t_1 = t_2 \quad t_2 = t_3}{t_1 = t_3} \qquad \frac{t = t'}{\sum_v t = \sum_v t'}$$

Table 2.2: Generalised Equational Logic

boxes. Only one of the boxes contains a prize, the other two do not. After choosing a box, the gameshow host opens one of the remaining two boxes, not containing the prize and again offers the candidate to choose. The candidate can either stick to his previous choice, or change his mind and select the other box. A typical pCRL specification of the game is found below.

$$\textbf{proc } \mathit{Game} = \mathit{choose} \cdot \mathit{remove} \cdot (\mathit{change} + \mathit{persist}) \cdot (\mathit{win} + \mathit{loose}) \tag{2.2}$$

Now, one may wonder whether the alternative specification, provided below, also models the same gameshow.

$$\textbf{proc } \mathit{Game}' = \mathit{choose} \cdot \mathit{remove} \cdot (\mathit{change} \cdot (\mathit{win} + \mathit{loose}) + \mathit{persist} \cdot (\mathit{win} + \mathit{loose})) \tag{2.3}$$

A simple calculation learns that on the basis of axiom A4, both specifications are essentially the same. However, a faulty model of the gameshow is the one, presented below.

$$\begin{aligned}\textbf{proc } \mathit{Game}'' = \;& \mathit{choose} \cdot \mathit{remove} \cdot \\ & ((\mathit{change} + \mathit{persist}) \cdot \mathit{win}) + ((\mathit{change} + \mathit{persist}) \cdot \mathit{loose})\end{aligned} \tag{2.4}$$

From this latter specification, we are able to deduce either two boxes contain a prize, or none of the boxes contains a prize. This does not fit in with the description of the gameshow. The law, using which we can relate the first and the last specifications is the left-distributivity law $x \cdot (y + z) = x \cdot y + x \cdot z$, which is complementary to axiom A4. This law is included in several theories, however, it is not part of ACP-centred process algebras.

Example 2.2.7. The sum-operator is a useful and powerful, yet sometimes hard to understand operator. As an example of the use of the sum-operator, we consider a toy-elevator system. The elevator, operating in a ten-storey building, can move up and down on request. Once it is occupied, it halts no sooner than the moment it has reached its destination. A specification of the elevator is given by (2.5).

$$\begin{aligned}&\textbf{proc } \mathit{Elevator}(s{:}\mathbb{N}, b{:}\mathbb{B}) = \\ &\quad \textstyle\sum_{n:\mathbb{N}} \mathit{request}(n) \cdot \mathit{Elevator}(n, \mathsf{t}) \lhd (n < s \lor s < n \leq 9) \land \neg b \rhd \delta \\ &\quad + \mathit{open} \cdot \mathit{Elevator}(s, \mathsf{f}) \lhd b \rhd \delta\end{aligned} \tag{2.5}$$

In this case, the sum-operator is used to specify the elevator can take any natural number as the requested destination. However, since the elevator operates in a ten-storey building, the only viable options are numbers, less than ten and different from the current storey.

The axiom system can be used to obtain an intuition behind the operators of pCRL; an operational perspective to the operators is obtained by defining their characteristics using an operational semantics. We define the semantics by associating a *Labelled Transition System* (LTS) to each process term of the language. We use Plotkin-style rules [101] to define a transition relation $\rightarrow$ of the LTS. We introduce the unary predicate $\rightarrow\surd$, denoting successful termination upon executing a transition.

For most languages, we can define the semantics by mapping the language constructs immediately onto the model. Unfortunately, this is tricky in the presence of (uninterpreted) data. We therefore introduce an intermediate step, needed for interpreting the uninterpreted process terms. The image is called the set of *processes* and is denoted by $\mathcal{P}$. Likewise, the set of interpretation of action terms is denoted $\mathcal{A}$, and referred to as the set of *actions*. The transition relation $\rightarrow \subseteq \mathcal{P} \times \mathcal{A} \times \mathcal{P}$ is written as $\mathsf{p} \xrightarrow{\mathsf{a}} \mathsf{p}'$ for actions a and processes p, p', and stands for the process p that can evolve to the process p' upon executing a. Likewise, the termination predicate $\rightarrow\surd \subseteq \mathcal{P} \times \mathcal{A}$, is written as $\mathsf{p} \xrightarrow{\mathsf{a}} \surd$ for actions a and processes p, and stands for the process p that can successfully terminate upon executing action a.

Definition 2.2.8 (*Semantics of pCRL*).
We associate a labelled transition system $\mathcal{L} = \langle \mathcal{P}, \mathcal{A}, \rightarrow, \rightarrow\surd \rangle$ to each expression in the theory of pCRL. We first define the interpretation of action terms and process terms.

$$\mathcal{A} = \{\mathsf{a}(d_1, \ldots, d_n) \mid \mathsf{a}{:}s_1 \times \cdots \times s_n \in \mathit{Act}, d_i \in \mathcal{D}_{s_i}\} \tag{2.6}$$

As a convention we write $\mathsf{a} \in \mathcal{A}$ when we are not interested in the (possible) parameters a can have. The set $\mathcal{A} \cup \{\delta\}$ is denoted $\mathcal{A}_\delta$. We define the set of processes as the set $\mathcal{P} = \bigcup_{i=0}^{\omega} \mathcal{P}^i$, where $\mathcal{P}^n$ is defined inductively by Eqn. (2.7).

$$\begin{array}{ll} \mathcal{P}^0 & = \mathcal{A}_\delta \\ \mathcal{P}^{n+1} & = \mathcal{P}^n \cup \{\mathsf{p} \cdot \mathsf{q}, \sum \mathsf{P} \mid \mathsf{p}, \mathsf{q} \in \mathcal{P}^n, \emptyset \subset \mathsf{P} \subseteq \mathcal{P}^n\} \end{array} \tag{2.7}$$

Note that we write $\sum \mathsf{P}$, for $\mathsf{P} \subseteq \mathcal{P}^n$ to denote the alternative composition of all elements in P. This construct should not be confused with the construct $\sum_v p$ for some process term p. The notation $\sum \mathsf{P}$ is first introduced in [50], and was subsequently adopted in [84, 108, 127]. For a motivation for introducing this construct, we refer to [50, 84].
Let ν be a data valuation. Expressions of pCRL are interpreted according to the interpretation function defined in Eqn. (2.8).

$$\begin{array}{ll} [\![\mathsf{a}(d_1, \ldots, d_n)]\!]^\nu & = \mathsf{a}([\![d_1]\!]^\nu, \ldots, [\![d_n]\!]^\nu) \\ [\![\delta]\!]^\nu & = \delta \\ [\![p + q]\!]^\nu & = \sum\{[\![p]\!]^\nu, [\![q]\!]^\nu\} \\ [\![p \cdot q]\!]^\nu & = [\![p]\!]^\nu \cdot [\![q]\!]^\nu \\ [\![\sum_v p]\!]^\nu & = \sum\{[\![p]\!]^{\nu'} \mid \nu(v') = \nu'(v') \text{ for all } v' \neq v\} \\ [\![p \lhd b \rhd q]\!]^\nu & = \begin{cases} [\![p]\!]^\nu & \text{if } [\![b]\!]^\nu \text{ is true} \\ [\![q]\!]^\nu & \text{if } [\![b]\!]^\nu \text{ is false} \end{cases} \end{array} \tag{2.8}$$

$$\mathsf{a} \xrightarrow{\mathsf{a}} \surd \qquad \frac{\mathsf{p} \xrightarrow{\mathsf{a}} \surd}{\sum(\mathsf{P} \cup \{\mathsf{p}\}) \xrightarrow{\mathsf{a}} \surd}$$

Table 2.3: Successful termination for pCRL, where $\mathsf{a} \in \mathcal{A}$, $\mathsf{p} \in \mathcal{P}$, $\mathsf{P} \subseteq \mathcal{P}$

$$\frac{\mathsf{p} \xrightarrow{\mathsf{a}} \surd}{\mathsf{p} \cdot \mathsf{q} \xrightarrow{\mathsf{a}} \mathsf{q}} \qquad \frac{\mathsf{p} \xrightarrow{\mathsf{a}} \mathsf{p}'}{\mathsf{p} \cdot \mathsf{q} \xrightarrow{\mathsf{a}} \mathsf{p}' \cdot \mathsf{q}} \qquad \frac{\mathsf{p} \xrightarrow{\mathsf{a}} \mathsf{p}'}{\sum(\mathsf{P} \cup \{\mathsf{p}\}) \xrightarrow{\mathsf{a}} \mathsf{p}'}$$

Table 2.4: Transition relation for pCRL, where $\mathsf{a} \in \mathcal{A}$, $\mathsf{p}, \mathsf{q}, \mathsf{p}' \in \mathcal{P}$, $\mathsf{P} \subseteq \mathcal{P}$

here, $\mathsf{a}{:}s_1 \times \cdots \times s_n \in$ *Act*, $p, q \in T_{\mathbb{P}}$ are process-closed terms, $b \in T_{\mathbb{B}}$ and $d_i \in T_{s_i}$. The deduction rules for successful termination and the transition relation are given in Tables (2.3) and (2.4).

The semantic equivalence we use in this thesis is induced by strong bisimulation. We first define the notion of simulation.

Definition 2.2.9 (*Simulation Relation for pCRL*).
A relation $\mathcal{R} \subseteq \mathcal{P} \times \mathcal{P}$ on processes is a simulation relation iff for all processes p and q, the following two properties are satisfied.

1. if $\mathsf{p}\mathcal{R}\mathsf{q}$ and $\mathsf{p} \xrightarrow{\mathsf{a}} \mathsf{p}'$ for some $\mathsf{a} \in \mathcal{A}$, then there exists a term q', such that $\mathsf{q} \xrightarrow{\mathsf{a}} \mathsf{q}'$ and $\mathsf{p}'\mathcal{R}\mathsf{q}'$.
2. if $\mathsf{p}\mathcal{R}\mathsf{q}$ and $\mathsf{p} \xrightarrow{\mathsf{a}} \surd$ for some $\mathsf{a} \in \mathcal{A}$, then also $\mathsf{q} \xrightarrow{\mathsf{a}} \surd$.

Definition 2.2.10 (*Strong Bisimulation for pCRL*).
A symmetric simulation relation $\mathcal{R} \subseteq \mathcal{P} \times \mathcal{P}$ on processes is called a strong bisimulation. Two processes p and q are strongly bisimilar, denoted by $\mathsf{p} \leftrightarrow \mathsf{q}$, iff there exists a strong bisimulation relation $\mathcal{R}$, such that $\mathsf{p}\mathcal{R}\mathsf{q}$.

Corollary 2.2.11 The union of all strong bisimulation relations induces an equivalence relation, generally referred to as *strong bisimilarity*.

Lemma 2.2.12. Strong bisimilarity is a congruence with respect to the operators defined on $\mathcal{P}$.

Proof. See [50, 84]. □

The interpretation of pCRL terms into labelled transition systems forms, together with bisimilarity, a model for the theory of pCRL. This model is referred to as the *bisimulation model* of pCRL.

Theorem 2.2.13 (*Soundness of pCRL*).
The theory pCRL is sound with respect to strong bisimulation, i.e. for all closed terms p and q, we have

$$\text{If } p = q \text{ then also for all data valuations } \nu\text{: } [\![p]\!]^{\nu} \leftrightarrow [\![q]\!]^{\nu} \tag{2.9}$$

Proof. See [84]. □

The soundness result means that for all process terms we can prove equivalent using the axiom system, a bisimulation relation exists. Notice that the converse does in general not hold for all algebras. However, Groote and Luttik showed that for pCRL, we can claim the converse, provided a number of requirements are fulfilled.

Theorem 2.2.14 (*Completeness of pCRL*).
The theory pCRL is relatively complete with respect to the operational semantics modulo strong bisimilarity, i.e. provided that the conditions below hold, then for all closed terms p and q, identity (2.10) holds.

1. The data algebra has an ω-*complete algebraic specification*,
2. The data-algebra has *built-in equality*,
3. The data-algebra has *built-in Skolem functions*.

   If for all data valuations ν we have $[\![p]\!]^\nu \leftrightarrow [\![q]\!]^\nu$ then also $p = q$ (2.10)

Proof. See the proofs in e.g. [50, 84] □

2.3 Syntax and Semantics of pCRL$_t$

The theory of pCRL, introduced in the previous section, is a small, yet elegant theory. It has been used as a vehicle for many case-studies, and is the subject of investigations into the theory of data-dependent systems. In this section, we introduce an extension of pCRL, adding primitives for dealing with quantitative timing information to the language. The resulting theory is called pCRL$_t$. We first address some background information on its design.

The design of the language pCRL$_t$ was guided by several desires. The three major considerations are listed below (see also [108]):

1. Every pCRL$_t$ specification, not referencing time, should appear exactly the same as a pCRL specification. Apart from their appearance, their intended behaviour should also be strongly related. Technically, the language pCRL$_t$ should be a *conservative extension* of the language pCRL. This implies the theory of pCRL$_t$ supersedes the theory of pCRL and, rather than investigating pCRL, we can study pCRL$_t$,
2. The extensions with time should be natural and concise,
3. The definition of the theory of pCRL$_t$ should be such that it supports the adaptation of many of the existing proof techniques, developed for pCRL to the timed setting.

Unlike many other languages, in pCRL$_t$, the time-domain is not fixed. However, as we already mentioned in Section 2.1, the time-domain of choice should at least be equipped with a total ordering $\leq$ and a least element with respect to this ordering, here denoted $\mathbf{0}$. Standard choices for the time-domain, such as the natural numbers $\mathbb{N}$, or the non-negative reals $\mathbb{R}_{\geq \mathbf{0}}$ can be used, but more exotic time-domains such as the singular set $\{\mathbf{0}\}$ are also allowed. In this chapter, we postpone the choice for a time-domain and use an arbitrary time-domain $\mathbb{T}$. The time-domain of choice for the rest of this thesis is $\mathbb{R}_{\geq \mathbf{0}}$.

The mechanism by which timing requirements are enforced in $p\text{CRL}_t$ is so-called *time-stamping* of processes. The building blocks of $p\text{CRL}_t$ consist of the operators and constants of pCRL, and two operators for expressing the timing requirements.

Definition 2.3.1 (*Signature of $p\text{CRL}_t$*).
The signature of the theory $p\text{CRL}_t$ consists of both a data signature (including the sort $\mathbb{T}$) and a process signature. We restrict our attention to the process part of $p\text{CRL}_t$. This part consists of the sort symbol $\mathbb{P}$, representing the set of process terms, the function declarations for pCRL, and the below function declarations:

1. *at operator* $\cdot\!\!:\mathbb{P} \times \mathbb{T} \to \mathbb{P}$. The process term $p{\cdot}t$ is intended as the process term p, restricted to the part of p that can start exactly at time t, if possible,
2. *initialisation operator* $\gg\,:\mathbb{T} \times \mathbb{P} \to \mathbb{P}$. The process $t \gg p$ is intended as the part of the process term p that can wait at least until time t.

Note that $p\text{CRL}_t$ employs a notion of *absolute timing* rather than *relative timing*. This means that time is measured from the start of the system, rather than from the moment the last action is executed.

Remark 2.3.2. *The choice for absolute timing in $p\text{CRL}_t$ is in some sense immaterial, since relative and absolute notions of time can easily be converted to each other without loss of information [14, 49, 15, 16]. Note that in [127], versions of pCRL with absolute timing and relative timing are discussed and proved sound and complete. In general, there are reasons to prefer the notion of absolute timing over relative timing: amongst others, absolute timing combines easier with parallelism, as is explained in [49, 108]. Note that many specification languages use the notion of relative timing, as it is in general easier to use; moreover, in a setting without additional features such as initialisation, timing inconsistencies will not arise with relative timing (such inconsistencies can possibly arise in the case of absolute timing [17]). As we shall see in Example 2.3.7, the absolute notion of time in $p\text{CRL}_t$ still permits us to mimic the notion of relative timing.*

Remark 2.3.3. *The time-stamping operator $\cdot$ is unique to $p\text{CRL}_t$ and μCRL_t, as it works on entire processes rather than on single actions as in e.g. [12, 15, 104]. The initialisation operator can also be found in e.g. [13, 73, 15].*

The extension of pCRL with the above timing constructs, has several side effects. Firstly, the constant δ no longer represents *inaction*, as it does allow for the passing of time. Hence, a more suitable name is *livelock*. Secondly, the process term $\delta{\cdot}t$ stands for a *time-lock* at time t: idling is allowed up to, and including time t, but no longer. Such a time-lock, occurring at time $\mathbf{0}$, is the timed counterpart of inaction in the untimed theory, as it represents no activity is possible (i.e. not even idling beyond the starting time). Thirdly, not all of the axioms, listed in Table 2.1, are valid anymore. For instance, a well-known consequence is the invalidation of the identity $x + \delta = x$. And rightfully so, for if we substitute the term $\delta{\cdot}\mathbf{0}$ for x, this would time-lock at time $\mathbf{0}$, rather than livelock. Notice that for any untimed x, the identity still holds.

We often omit parentheses and use the binding strength given by Eqn. (2.11), where we use $\geq$ to denote the lefthand-side operator binds stronger than the righthand-side operator.

$$\text{‹} \;\geq\; \cdot \;\geq \gg \geq\; \triangleleft\,\triangleright \;\geq\; \sum_v \;\geq\; + \tag{2.11}$$

We first list the axioms for pCRL$_t$ and subsequently provide some example pCRL$_t$ specifications and a brief explanation of some of the axioms.

Definition 2.3.4 (*Axiomatisation of pCRL$_t$*).
The axioms for pCRL$_t$ are the axioms A1 through A5, A7, SUM1, SUM3 through SUM5, C1 through C4 and C6 and C7 of pCRL, supplemented by the axioms listed in Table 2.5.

A6$^-$	$a+\delta$	$=$	a	
A6’	$x+\delta\text{‹}\mathbf{0}$	$=$	x	
PE’	$p \triangleleft \mathit{eq}(v,w) \triangleright \delta\text{‹}\mathbf{0}$	$=$	$p[v/w] \triangleleft \mathit{eq}(v,w) \triangleright \delta\text{‹}\mathbf{0}$	
AT1	x	$=$	$\sum_t x\text{‹}t$	when $t \notin FV(x)$
AT2	$a\text{‹}t \cdot x$	$=$	$a\text{‹}t \cdot (t \gg x)$	
ATA1	$a\text{‹}t\text{‹}u$	$=$	$(a\text{‹}t \triangleleft u \leq t \triangleright \delta\text{‹}t) \triangleleft t \leq u \triangleright \delta\text{‹}u$	
ATA2	$(x+y)\text{‹}t$	$=$	$x\text{‹}t + y\text{‹}t$	
ATA3	$(x \cdot y)\text{‹}t$	$=$	$x\text{‹}t \cdot y$	
ATA4	$(\sum_v p)\text{‹}t$	$=$	$\sum_v p\text{‹}t$	when $v \notin FV(t)$
ATA5	$(x \triangleleft b \triangleright y)\text{‹}t$	$=$	$x\text{‹}t \triangleleft b \triangleright y\text{‹}t$	
ATB1	$t \gg a\text{‹}u$	$=$	$a\text{‹}u \triangleleft t \leq u \triangleright \delta\text{‹}t$	
ATB2	$t \gg (x+y)$	$=$	$t \gg x + t \gg y$	
ATB3	$t \gg (x \cdot y)$	$=$	$(t \gg x) \cdot y$	
ATB4	$t \gg (x \triangleleft b \triangleright y)$	$=$	$(t \gg x) \triangleleft b \triangleright (t \gg y)$	
C3’	$x \triangleleft b \triangleright y$	$=$	$x \triangleleft b \triangleright \delta\text{‹}\mathbf{0} + y \triangleleft \neg b \triangleright \delta\text{‹}\mathbf{0}$	
C5’	$x \triangleleft b_1 \triangleright \delta\text{‹}\mathbf{0} + x \triangleleft b_2 \triangleright \delta\text{‹}\mathbf{0}$	$=$	$x \triangleleft b_1 \vee b_2 \triangleright \delta\text{‹}\mathbf{0}$	
SCA’	$(x \triangleleft b \triangleright \delta\text{‹}\mathbf{0}) \cdot (y \triangleleft b \triangleright \delta\text{‹}\mathbf{0})$	$=$	$x \cdot y \triangleleft b \triangleright \delta\text{‹}\mathbf{0}$	
SUM12’	$(\sum_v p) \triangleleft b \triangleright \delta\text{‹}\mathbf{0}$	$=$	$\sum_v p \triangleleft b \triangleright \delta\text{‹}\mathbf{0}$	when $v \notin FV(b)$

Table 2.5: Axioms for pCRL$_t$, where $x,y,z \in V_{\mathbb{P}}$, process-closed $p,q \in T_{\mathbb{P}}$, $a \in AT_\delta$, $t,u \in V_{\mathbb{T}}$, $b,b_1,b_2 \in V_{\mathbb{B}}$ and $v,w \in V$.

The axiom-system of pCRL$_t$ is in most respects a straightforward extension of pCRL. Most notable changes are the replacements of the untimed δ of pCRL to its true timed counterpart $\delta\text{‹}\mathbf{0}$. This is the case in e.g. the axioms C3’ and SUM12’, but the largest effect is the rephrased axiom A6, yielding axioms A6’ and A6$^-$.

Remark 2.3.5. *In case the conditional operator has the form $p \lhd b \rhd \delta{}^{\backprime}\mathbf{0}$, we sometimes use the convention to write $[b] :: \!\!\rightarrow p$. Notice that we thereby overload the definition of $:: \!\!\rightarrow$. From the context in which it is used, however, it is clear whether the operator $:: \!\!\rightarrow$ is used in a timed setting or an untimed setting.*

The embedding of the untimed theory into the timed theory is most apparent in the axioms AT1, AT2 and ATA1. From axiom AT1, we can basically deduce that an arbitrary (timed or untimed) process really is a process that can be executed at *any time*. It may be clear this can lead to inconsistencies if we do not carefully specify what is meant by e.g. a single action a that must occur simultaneously at two (possibly different) moments of time. This situation is dealt with by axiom ATA1. Axiom AT2 encodes the way we perceive the natural flow of time: unidirectional, meaning that once an action at a specific time has been executed, successive actions cannot occur before this moment in time. Notice that we can specify processes in which two successive actions happen at the same moment in time, e.g. $a{}^{\backprime}3 \cdot b{}^{\backprime}3$ means action a is executed at time 3, followed by action b at time 3. This phenomenon is called *urgency* of actions.

The at-operator is applied to entire processes. Yet, from axioms ATA2 through ATA5 we can deduce it propagates to the first actions of a process, and leaves the other parts of the process virtually unaffected. It is in the combination of the at-operator, the sum-operator and the conditionals that the theory of pCRL$_t$ finds its strength. This allows us to specify arbitrary intervals, sets, etc. as execution times for a process.

Example 2.3.6. Using the at-operator, the sum-operator and the conditionals, we can specify a drifting clock, i.e. a clock that is accurate within a bounded interval $[1-\eth, 1+\eth]$ for $\eth < 1$.

$$\textbf{proc } \mathit{Clock}(t{:}\mathbb{T}) = \sum_{\varepsilon:\mathbb{T}} \mathit{tick}{}^{\backprime}(t+\varepsilon) \cdot \mathit{Clock}(t+\varepsilon) \lhd \varepsilon \in [1-\eth, 1+\eth] \rhd \delta{}^{\backprime}\mathbf{0} \tag{2.12}$$

The language pCRL$_t$ is based on the notion of absolute time. This means that in our time-stamps we refer to absolute moments in time. In some instances, however, it is easier to express timing, relative to the occurrence of some event, i.e. use relative time rather than absolute time. Again, using the combination of the at-operator, the sum-operator and the conditionals, we can express relative timing.

Example 2.3.7. We consider a time-critical application: a coffee-vending machine for high-priced managers. The machine issues, upon the insertion of a coin, either espresso or cappuccino, depending on the manager's choice. The machine operates according to the credo "time = money" and, hence, issues espresso within two seconds after its request and cappuccino within three seconds after its request.

$$\textbf{proc } \mathit{Machine}_{\mathit{idle}}(t{:}\mathbb{T}) = \textstyle\sum_{u:\mathbb{T}} \mathit{coin}{}^{\backprime}u \cdot \mathit{Machine}_{\mathit{busy}}(u)$$

$$\begin{aligned}
&\textbf{proc } \mathit{Machine}_{\mathit{busy}}(t{:}\mathbb{T}) = \\
&\quad \textstyle\sum_{v:\mathbb{T}} (\mathit{espresso}{}^{\backprime}v \cdot (\sum_{d:\mathbb{T}} \mathit{issue}_e{}^{\backprime}(v+d) \cdot \mathit{Machine}_{\mathit{idle}}(v+d) \lhd d \leq 2 \rhd \delta{}^{\backprime}\mathbf{0}) \\
&\qquad + \mathit{cappuccino}{}^{\backprime}v \cdot (\textstyle\sum_{d:\mathbb{T}} \mathit{issue}_c{}^{\backprime}(v+d) \cdot \mathit{Machine}_{\mathit{idle}}(v+d) \lhd d \leq 3 \rhd \delta{}^{\backprime}\mathbf{0}))
\end{aligned} \tag{2.13}$$

Using the sum-operator, we have thus specified a relative timing dependency on e.g. the issue_e action and the action *espresso*. This feature makes pCRL$_t$ a very flexible language.

The semantics of $p\mathrm{CRL}_t$ is defined in much the same way as we defined the semantics for $p\mathrm{CRL}$. Obviously, ordinary labelled transition systems will not do in this situation, as this model does not adequately capture timing information. Hence, we define an operational semantics for each $p\mathrm{CRL}_t$ expression by associating a *Timed Labelled Transition Systems* to each expression. For this, we use a *Time-Stamped Labelled Transition System* (TSLTS), see e.g. [73]. In a TSLTS, we distinguish two concepts: *timed action transitions* and a *delay predicate*. A timed action transition is a transition relation to which timing information is added. The delay predicate provides information on the maximal time a process, represented by a TSLTS, can stay in a particular state. We again use Plotkin-style rules [101] to define the transition relation and the delay predicate of the TSLTS.

Remark 2.3.8. *Note that in [49], the semantics of $p\mathrm{CRL}_t$ is defined in terms of a Two-Phase Labelled Transition System (TPLTS). In a TPLTS, the passage of time is represented by a labelled transition of one state to another state, where the label contains a (positive) number, representing the amount of time that has passed in taking the transition. It has been shown that under certain conditions, both transition systems can be converted to each other (see e.g. [73]). The paper [108] we use to base our presentation of $p\mathrm{CRL}_t$ on defines its semantics in terms of TSLTSs.*

An intermediate interpretation is again needed to interpret data and timing variables. We extend the set of processes $\mathcal{P}$ to incorporate the quantitative timing-dependencies. The set of interpreted action terms is unaffected by the addition of time.

Definition 2.3.9 (*Semantics of $p\mathrm{CRL}_t$*).
We associate a time-stamped labelled transition system $\mathcal{L} = \langle \mathcal{P}, \mathcal{A}, \rightarrow, \rightarrow\surd, \mathcal{U} \rangle$ to each expression in the theory of $p\mathrm{CRL}_t$. We first redefine the set of processes $\mathcal{P} = \bigcup_{i=0}^{\omega} \mathcal{P}^i$, where $\mathcal{P}^n$ is defined inductively by Eqn. (2.14).

$$\begin{array}{ll} \mathcal{P}^0 & = \mathcal{A}_\delta \\ \mathcal{P}^{n+1} & = \mathcal{P}^n \cup \{\mathsf{p} \cdot \mathsf{q}, \sum \mathsf{P}, \mathsf{p} \lhd t, t \gg \mathsf{p} \mid \mathsf{p}, \mathsf{q} \in \mathcal{P}^n, \emptyset \subset \mathsf{P} \subseteq \mathcal{P}^n, t \in \mathcal{D}_{\mathbb{T}}\} \end{array} \tag{2.14}$$

Expressions of $p\mathrm{CRL}_t$ are interpreted according to the interpretation function we defined in Eqn (2.8), extended with the identities of (2.15), where ν is a valuation for the data and time variables.

$$\begin{array}{ll} [\![t \gg p]\!]^\nu & = [\![t]\!]^\nu \gg [\![p]\!]^\nu \\ [\![p \lhd t]\!]^\nu & = [\![p]\!]^\nu \lhd [\![t]\!]^\nu \end{array} \tag{2.15}$$

here, $p \in T_{\mathbb{P}}$ and $t \in T_{\mathbb{T}}$. The deduction rules for successful termination, the transition relation and the ultimate delay are listed in Tables 2.6, 2.7 and 2.8.

$$\mathsf{a} \xrightarrow{\mathsf{a}}_t \surd \qquad \frac{\mathsf{p} \xrightarrow{\mathsf{a}}_t \surd}{\mathsf{p} \lhd t \xrightarrow{\mathsf{a}}_t \surd} \qquad \frac{\mathsf{p} \xrightarrow{\mathsf{a}}_t \surd}{\sum(\mathsf{P} \cup \{\mathsf{p}\}) \xrightarrow{\mathsf{a}}_t \surd} \qquad \frac{\mathsf{p} \xrightarrow{\mathsf{a}}_t \surd \quad t' \leq t}{t' \gg \mathsf{p} \xrightarrow{\mathsf{a}}_t \surd}$$

Table 2.6: Successful termination for $p\mathrm{CRL}_t$, where $\mathsf{a} \in \mathcal{A}$, $\mathsf{p} \in \mathcal{P}$, $t, t' \in \mathcal{D}_{\mathbb{T}}$ and $\mathsf{P} \subseteq \mathcal{P}$.

$$
\frac{\mathsf{p} \xrightarrow{\mathsf{a}}_t \mathsf{p}'}{\mathsf{p}{}^{\varsigma}t \xrightarrow{\mathsf{a}}_t \mathsf{p}'}
\qquad
\frac{\mathsf{p} \xrightarrow{\mathsf{a}}_t \mathsf{p}'}{\sum(\mathsf{P} \cup \{\mathsf{p}\}) \xrightarrow{\mathsf{a}}_t \mathsf{p}'}
\qquad
\frac{\mathsf{p} \xrightarrow{\mathsf{a}}_t \mathsf{p}'}{\mathsf{p} \cdot \mathsf{q} \xrightarrow{\mathsf{a}}_t \mathsf{p}' \cdot \mathsf{q}}
$$

$$
\frac{\mathsf{p} \xrightarrow{\mathsf{a}}_t \mathsf{p}' \quad t' \leq t}{t' \gg \mathsf{p} \xrightarrow{\mathsf{a}}_t \mathsf{p}'}
\qquad
\frac{\mathsf{p} \xrightarrow{\mathsf{a}}_t \surd}{\mathsf{p} \cdot \mathsf{q} \xrightarrow{\mathsf{a}}_t t \gg \mathsf{q}}
$$

Table 2.7: Transition relation for pCRL$_t$, where $\mathsf{a} \in \mathcal{A}$, $\mathsf{p}, \mathsf{q}, \mathsf{p}' \in \mathcal{P}$, $t, t' \in \mathcal{D}_{\mathbb{T}}$ and $\mathsf{P} \subseteq \mathcal{P}$.

$$
\mathcal{U}_t(a)
\qquad
\frac{\mathcal{U}_t(\mathsf{p})}{\mathcal{U}_t(\mathsf{p} \cdot \mathsf{q})}
\qquad
\frac{\mathcal{U}_t(\mathsf{p})}{\mathcal{U}_t(\sum(\mathsf{P} \cup \{\mathsf{p}\}))}
\qquad
\frac{\mathcal{U}_t(\mathsf{p}) \quad t \leq t'}{\mathcal{U}_t(\mathsf{p}{}^{\varsigma}t')}
\qquad
\frac{\mathcal{U}_t(\mathsf{p})}{\mathcal{U}_t(t' \gg \mathsf{p})}
\qquad
\frac{t \leq t'}{\mathcal{U}_t(t' \gg \mathsf{p})}
$$

Table 2.8: Deduction rules for the delay relation for pCRL$_t$, where $\mathsf{a} \in \mathcal{A}$, $\mathsf{p}, \mathsf{q}, \mathsf{p}' \in \mathcal{P}$, $t, t' \in \mathcal{D}_{\mathbb{T}}$ and $\mathsf{P} \subseteq \mathcal{P}$.

We adapt the notion of equality for pCRL to a timed setting in the usual manner. The timed version of strong bisimulation is called *strong timed bisimulation*

Definition 2.3.10 (*Strong Timed Bisimulation for pCRL$_t$*).
A symmetric relation $\mathcal{R} \subseteq \mathcal{P} \times \mathcal{P}$ on processes is a strong timed bisimulation iff for all processes p and q, the following three properties are satisfied.

1. if $\mathsf{p}\mathcal{R}\mathsf{q}$ and $\mathsf{p} \xrightarrow{\mathsf{a}}_t \mathsf{p}'$ for some $\mathsf{a} \in \mathcal{A}$ and $t \in \mathcal{D}_{\mathbb{T}}$, then there exists a process q', such that $\mathsf{q} \xrightarrow{\mathsf{a}}_t \mathsf{q}'$ and $\mathsf{p}'\mathcal{R}\mathsf{q}'$ holds.

2. if $\mathsf{p}\mathcal{R}\mathsf{q}$ and $\mathsf{p} \xrightarrow{\mathsf{a}}_t \surd$ for some $\mathsf{a} \in \mathcal{A}$ and $t \in \mathcal{D}_{\mathbb{T}}$, then also $\mathsf{q} \xrightarrow{\mathsf{a}}_t \surd$.

3. if $\mathsf{p}\mathcal{R}\mathsf{q}$, then $\mathcal{U}_t(\mathsf{p})$ iff $\mathcal{U}_t(\mathsf{q})$ for all $t \in \mathcal{D}_{\mathbb{T}}$.

Two pCRL$_t$ processes p and q are strong timed bisimilar, denoted by $\mathsf{p} \underline{\leftrightarrow}_t \mathsf{q}$, iff there exists a strong timed bisimulation relation $\mathcal{R}$, such that $\mathsf{p}\mathcal{R}\mathsf{q}$.

Corollary 2.3.11 The union of all strong timed bisimulation relations induces an equivalence relation on timed processes, generally referred to as *strong timed bisimilarity*.

Lemma 2.3.12. Strong timed bisimilarity is a congruence with respect to the operators defined on $\mathcal{P}$.

Proof. See [108]. □

The interpretation of pCRL$_t$ terms into timed labelled transition systems forms, together with timed bisimilarity, a model for the theory of pCRL$_t$. This model is referred to as the *strong timed bisimulation model* of pCRL$_t$.

Theorem 2.3.13 (*Soundness of pCRL$_t$*).
The theory of pCRL$_t$ is sound with respect to strong timed bisimulation, i.e. for all closed terms p and q and all data valuations ν, we have

$$\text{If } p = q \text{ then also for all data valuations } \nu\text{: } [\![p]\!]^\nu \leftrightarrows_t [\![q]\!]^\nu \tag{2.16}$$

Proof. See [108]. □

Theorem 2.3.14 (*Completeness of pCRL$_t$*).
The theory pCRL$_t$ is relatively complete with respect to the operational semantics modulo strong timed bisimilarity, i.e. provided that the conditions, listed in Theorem 2.2.14 hold, then for all closed terms p and q, identity (2.17) holds.

$$\text{If for all data valuations } \nu\text{, we have } [\![p]\!]^\nu \leftrightarrows_t [\![q]\!]^\nu \text{ then also } p = q \tag{2.17}$$

Proof. See [108] for an extensive treatment. □

2.4 Syntax and Semantics of μCRL$_t$

The expressivity of the language pCRL$_t$ is sufficient for describing most real-life systems up to a minute accuracy. It moreover provides an elegant theory for the investigations of the interplay of time, arbitrary data and discrete actions. However, the fact that we are restricted to the use of sequential and alternative composition (or quantification), seriously limits the practical applicability of the language. For this reason, pCRL$_t$ has been extended with constructs for dealing with parallelism, yielding the language μCRL$_t$. Note that μCRL$_t$ is a conservative extension of μCRL.

Concurrency is dealt with by adding new operators to the language pCRL$_t$. The building blocks of μCRL$_t$ therefore consist of the operators and constants of pCRL$_t$, four operators for capturing concurrent behaviour and two additional operators that are useful in the context of concurrency.

Definition 2.4.1 (*Signature of μCRL$_t$*).
The signature of the theory μCRL$_t$ consists of both a data signature (including the sort $\mathbb{T}$) and a process signature. We restrict our attention to the process part of μCRL$_t$. This part consists of the sort symbol $\mathbb{P}$, representing the set of process terms, the function declarations for pCRL$_t$, and the function declarations given below:

1. *parallel operator* $\|{:}\mathbb{P} \times \mathbb{P} \to \mathbb{P}$. The process term $p\|q$ is intended to represent the interleavings of process terms p and q,

2. *left-merge operator* $⌊\!⌊{:}\mathbb{P} \times \mathbb{P} \to \mathbb{P}$. The process term $p ⌊\!⌊\, q$ is intended to represent the interleavings of process terms p and q with the assumption that the first action originates from process term p,

3. *communication merge* $|{:}\mathbb{P} \times \mathbb{P} \to \mathbb{P}$. The process term $p|q$ is intended to represent the interleavings of process terms p and q with the assumption that the first action is a result from a synchronisation between process terms p and q. The action resulting from such a

communication is defined by the binary, commutative and associative function γ which is only defined on *action declarations*. In order for a communication to occur between action terms $\mathsf{a}(d), \mathsf{a}'(e) \in AT$, $\gamma(\mathsf{a}, \mathsf{a}')$ should be defined, and the data parameters d and e should match,

4. *before operator* $\ll :\mathbb{P} \times \mathbb{P} \to \mathbb{P}$. The process term $p \ll q$ represents the process term p restricted to the part of p that can start before process term q gets definitely disabled.

5. *encapsulation operator* $\partial_H{:}\mathbb{P} \to \mathbb{P}$. The process term $\partial_H(p)$, where $H \subseteq$ *Act*, represents the process term p in which all occurrences of actions $\mathsf{a} \in H$, have been renamed to δ. This operator is useful in enforcing synchronisations, by blocking autonomous behaviour.

6. *renaming operator* $\rho_R{:}\mathbb{P} \to \mathbb{P}$. The process term $\rho_R(p)$, where $R{:}$*Act* $\to$ *Act*, represents the process term p in which all occurrences of actions $\mathsf{a} \in \mathsf{dom}(R)$ have been renamed to $R(\mathsf{a})$.

Most of the above-listed operators (in fact, with the exception of the *before* operator, *all* of the above-listed operators) are also part of μCRL. The renaming operator is, strictly speaking, not part of the theory μCRL$_t$ as studied in e.g. [49, 108], however, it is part of the theory μCRL. Since the renaming operator is used on various occasions in this thesis, we have added it to the theory μCRL$_t$ as presented here.

The binding strength of the μCRL$_t$ operators is listed in Eqn. (2.18), where we use $\geq$ to denote that the lefthand-side operator(s) binds stronger than the righthand-side operator(s).

$$ {}^{\varsigma} \;\geq\; \{\partial_H, \rho_R\} \;\geq\; \cdot \;\geq\; \{\gg, \ll\} \;\geq\; \{\lhd\,\rhd, \|, ⌊⌊, |\} \;\geq\; \sum_v \;\geq\; + \tag{2.18}$$

As can be gathered from the comments in the above definition, the formalism μCRL$_t$ is based on the paradigm of *interleavings* rather than on *true concurrency*. This choice originates from the desire to have a theory that extends on the untimed μCRL theory, and at the same time, suppresses the proliferation of the number of *concurrent* events, or *multi-actions* in calculations.

Definition 2.4.2 (*Axiomatisation of μCRL$_t$*).
The axioms for μCRL$_t$ are the axioms of pCRL$_t$, supplemented by the axioms, listed in Tables 2.9 and 2.10.

A brief glance at the axioms learns that many of the axioms for μCRL carry over immediately to the setting with time. The necessity of the (auxiliary) operator $\ll$ may come as a surprise. However, this operator is needed to express that two process terms p and q, in a parallel context, can execute actions exactly as long as the other process can wait. For example, the process $(a{}^{\varsigma}2 + b{}^{\varsigma}4) \| c{}^{\varsigma}3$ *cannot* choose to execute action $b{}^{\varsigma}4$ first, simply because the process $c{}^{\varsigma}3$ ceases to "exist" at time 3.

One may wonder why the encapsulation operator is needed. This operator has come forth from the desire to block unwanted actions. For instance, in a communication between two processes, three scenarios are possible: two scenarios due to interleaving (i.e. no communication has occurred!) and a scenario in which both parties agree on a communication. However, we are often interested in the result of the communications only. An effective way of disabling the redundant scenarios is by renaming their "starting action" to inaction.

CM1	$x\Vert y$	$=$	$x \mathbin{\vert\!\!\lfloor} y + y \mathbin{\vert\!\!\lfloor} x + x\vert y$
CM2	$a\text{‹}t \mathbin{\vert\!\!\lfloor} x$	$=$	$(a\text{‹}t \ll x) \cdot x$
CM3	$a\text{‹}t \cdot x \mathbin{\vert\!\!\lfloor} y$	$=$	$(a\text{‹}t \ll y) \cdot (t \gg x\Vert y)$
CM4	$(x+y) \mathbin{\vert\!\!\lfloor} z$	$=$	$x \mathbin{\vert\!\!\lfloor} z + y \mathbin{\vert\!\!\lfloor} z$
CM5	$a \cdot x\vert a'$	$=$	$(a\vert a') \cdot x$
CM6	$a\vert a' \cdot x$	$=$	$(a\vert a') \cdot x$
CM7	$a \cdot x\vert a' \cdot y$	$=$	$(a\vert a') \cdot (x\Vert y)$
CM8	$(x+y)\vert z$	$=$	$x\vert z + y\vert z$
CM9	$x\vert(y+z)$	$=$	$x\vert y + x\vert z$
H18	$(x \lhd b \rhd y) \mathbin{\vert\!\!\lfloor} z$	$=$	$(x \mathbin{\vert\!\!\lfloor} z) \lhd b \rhd (y \mathbin{\vert\!\!\lfloor} z)$
CF	$\mathsf{a}(\vec{d})\vert\mathsf{a}'(\vec{e})$	$=$	$\begin{cases} \gamma(\mathsf{a},\mathsf{a}')(\vec{d}) \lhd \boldsymbol{eq}(\vec{d},\vec{e}) \rhd \delta \\ \qquad \text{if } \gamma(\mathsf{a},\mathsf{a}') \text{ defined} \\ \delta \text{ otherwise} \end{cases}$
CD1	$\delta\vert a$	$=$	δ
CD2	$a\vert\delta$	$=$	δ
ATA7	$(x\vert y)\text{‹}t$	$=$	$x\text{‹}t\vert y$
ATA8	$(x\vert y)\text{‹}t$	$=$	$x\vert y\text{‹}t$
ATC1	$x \ll a\text{‹}t$	$=$	$\sum_u x\text{‹}u \lhd u \leq t \rhd x\text{‹}t$
ATC2	$x \ll (y+z)$	$=$	$x \ll y + x \ll z$
ATC3	$x \ll (y \cdot z)$	$=$	$x \ll y$
ATC4	$x \ll \sum_v p$	$=$	$\sum_v x \ll p$
ATC5	$x \ll (y \lhd b \rhd z)$	$=$	$(x \ll y) \lhd b \rhd (x \ll z)$
SUM6	$(\sum_v p) \mathbin{\vert\!\!\lfloor} x$	$=$	$\sum_v (p \mathbin{\vert\!\!\lfloor} x)$
SUM7	$(\sum_v p)\vert x$	$=$	$\sum_v (p\vert x)$
SUM7’	$x\vert(\sum_v p)$	$=$	$\sum_v (x\vert p)$

Table 2.9: Axioms for μCRL$_t$, where $x, y, z \in V_{\mathbb{P}}$, process-closed $p \in T_{\mathbb{P}}$, $a, a' \in AT_\delta$, $\mathsf{a}, \mathsf{a}' \in$ *Act*, $b \in T_{\mathbb{B}}$ and $v \in V$

Example 2.4.3. An analogy between parallelism in μCRL$_t$ and a real-life situation is the following. Consider two lanes that gradually merge to become a single lane (see Fig. 2.1). Suppose we have three observers (x, y and z) located at three different sections of the lanes. During a given time-interval, observer x first signals a car a and then a car b, whereas observer y only signals a car c. The order in which observer z sees cars passing by, can then be specified as a μCRL$_t$ expression. The observations of x are given by the equation X, the observations of y by

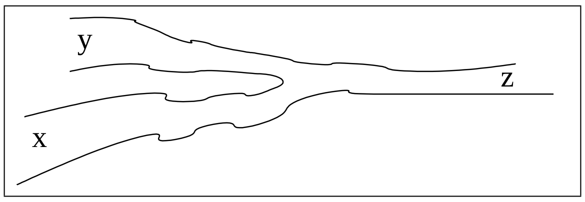


Figure 2.1: Traffic Lanes

$$\begin{array}{l} X = a \cdot b \\ Y = c \\ Z = X \| Y \end{array} \tag{2.19}$$

equation Y and the observations of z by Z. A communication of two cars in reality corresponds to a collision, hence, we define $\gamma(a, c) = \gamma(b, c) = \dagger$. The calculation below shows the possible observations z can make.

$$\begin{array}{rcl} Z & = & a \cdot b \| c \\ & = & \sum_{t,u,v} (a^{\varsigma}t \cdot b^{\varsigma}u \| c^{\varsigma}v) \\ & = & \sum_{t,u,v} ((a^{\varsigma}t \ll c^{\varsigma}v) \cdot (t \gg b^{\varsigma}u \| c^{\varsigma}v) + (c^{\varsigma}v \ll a^{\varsigma}t) \cdot (v \gg a^{\varsigma}t \cdot b^{\varsigma}u) + (\dagger^{\varsigma}t \cdot b^{\varsigma}u)) \\ & = & a \cdot \sum_{u,v} ((b^{\varsigma}u \ll c^{\varsigma}v) \cdot (u \gg c^{\varsigma}v) + (c^{\varsigma}v \ll b^{\varsigma}u) \cdot (v \gg b^{\varsigma}u) + (\dagger^{\varsigma}u)) \\ & & +c \cdot a \cdot b + \dagger \cdot b \\ & = & a \cdot (b \cdot c + c \cdot b + \dagger) + c \cdot a \cdot b + \dagger \cdot b \end{array} \tag{2.20}$$

If the observers x, y and z undertake the same activities, knowing there is a traffic-light regulation system, ensuring the safe merging of cars, the above situation is not entirely correct. In fact, observer z cannot see collisions. Hence, we change the equation for Z to $Z = \partial_{\{\dagger\}}(X \| Y)$. In effect, the observations for z can now be represented by the expression below.

$$Z \;=\; a \cdot (b \cdot c + c \cdot b) + c \cdot a \cdot b \tag{2.21}$$

In this example, we used the encapsulation operator to remove all scenarios preceded by a communication (the collision). However, in most cases, it is the communication that is desired rather than the interleavings.

The semantics of μCRL$_t$ is again defined using the model of time-stamped labelled transition systems. The process interpretations are extended with the new operators for μCRL$_t$.

Definition 2.4.4 (*Semantics of μCRL$_t$*).
We associate a time-stamped labelled transition system $\mathcal{L} = \langle \mathcal{P}, \mathcal{A}, \rightarrow, \rightarrow\surd, \mathcal{U} \rangle$ to each expression in the theory of μCRL$_t$. We redefine the set of processes $\mathcal{P} = \bigcup_{i=0}^{\omega} \mathcal{P}^i$, where $\mathcal{P}^n$ is defined inductively by Eqn. (2.22).

$$\begin{array}{ll} \mathcal{P}^0 & = \mathcal{A}_\delta \\ \mathcal{P}^{n+1} & = \mathcal{P}^n \cup \{\mathsf{p} \cdot \mathsf{q}, \sum \mathsf{P}, \mathsf{p} \| \mathsf{q}, \mathsf{p} \,\lfloor\!\lfloor\, \mathsf{q}, \mathsf{p} | \mathsf{q}, \mathsf{p}^{\varsigma}t, t \gg \mathsf{p}, \mathsf{p} \ll \mathsf{q}, \rho_R(\mathsf{p}), \partial_H(\mathsf{p}) \mid \\ & \qquad \mathsf{p}, \mathsf{q} \in \mathcal{P}^n, \emptyset \subset \mathsf{P} \subseteq \mathcal{P}^n, t \in \mathcal{D}_{\mathbb{T}}, H \subseteq \mathit{Act}, R{:}\mathit{Act} \rightarrow \mathit{Act}\} \end{array} \tag{2.22}$$

Expressions of μCRL$_t$ are interpreted according to the interpretation function we defined in Eqn. 2.15, extended with the identities of (2.23), where ν again is a valuation for the data and time variables.

$$\begin{array}{ll} [\![p \| q]\!]^\nu = & [\![p]\!]^\nu \| [\![q]\!]^\nu \\ [\![p \mathbin{\|\!\_} q]\!]^\nu = & [\![p]\!]^\nu \mathbin{\|\!\_} [\![q]\!]^\nu \\ [\![p | q]\!]^\nu = & [\![p]\!]^\nu | [\![q]\!]^\nu \\ [\![p \ll q]\!]^\nu = & [\![p]\!]^\nu \ll [\![q]\!]^\nu \\ [\![\partial_H(p)]\!]^\nu = & \partial_H([\![p]\!]^\nu) \\ [\![\rho_R(p)]\!]^\nu = & \rho_R([\![p]\!]^\nu) \end{array} \tag{2.23}$$

here, $p, q \in T_{\mathbb{P}}$ are process-closed terms, $H \subseteq$ *Act* is an encapsulation set and R:*Act* $\rightarrow$ *Act* is a relabelling. The deduction rules for successful timed termination, the transition relation and the ultimate delay are the rules for pCRL$_t$ supplemented with the rules listed in Tables 2.11, 2.12 and 2.13.

The notion of equivalence, used for pCRL$_t$, extends straightforwardly to μCRL$_t$. We therefore omit its definition, and refer to Def. 2.3.10.

Lemma 2.4.5. Strong timed bisimilarity is a congruence with respect to the operators defined on $\mathcal{P}$.

Proof. See [108]. □

Theorem 2.4.6 (*Soundness of μCRL$_t$*). The theory of μCRL$_t$ is sound with respect to strong timed bisimulation, i.e. for all closed terms p and q, we have

$$\text{If } p = q \text{ then also for all data valuations } \nu\text{: } [\![p]\!]^\nu \mathrel{\underline{\leftrightarrow}}_t [\![p]\!]^\nu \tag{2.24}$$

Proof. See [108]. □

Theorem 2.4.7 (*Completeness of μCRL$_t$*).
The theory μCRL$_t$ is relatively complete with respect to the operational semantics modulo timed strong bisimilarity, i.e. provided that the conditions, listed in Theorem 2.2.14 hold, then for all closed terms p and q, identity (2.25) holds.

$$\text{If for all data valuations } \nu\text{, we have } [\![p]\!]^\nu \mathrel{\underline{\leftrightarrow}}_t [\![q]\!]^\nu \text{ then also } p = q \tag{2.25}$$

Proof. We refer to [108] for an extensive proof. □

2.5 Normal Forms

The proof techniques, developed for μCRL and μCRL$_t$ mostly operate on the assumption that a process has been rewritten into a special format. This format is also used in the tool suite for μCRL, i.e. the μCRL-toolbox [126]. In [119], it is proved all guarded μCRL$_t$ (and therefore also μCRL) expressions can be rewritten to this special format.

The format, known as *Linear Process Equation* (LPE) [119], or *Linear Process Operator* (LPO), relies very heavily on the use of data in μCRL$_t$. It basically encodes the state space as a μCRL$_t$ data type and defines functions on this state space that represent the transitions between states. The format for an untimed LPE has the following appearance.

DD	$\partial_H(\delta)$	$=$	δ	
D1	$\partial_H(\mathsf{a}(\vec{d}))$	$=$	$\mathsf{a}(\vec{d})$	if $\mathsf{a} \notin H$
D2	$\partial_H(\mathsf{a}(\vec{d}))$	$=$	δ	if $\mathsf{a} \in H$
D3	$\partial_H(x+y)$	$=$	$\partial_H(x)+\partial_H(y)$	
D4	$\partial_H(x\cdot y)$	$=$	$\partial_H(x)\cdot\partial_H(y)$	
D5	$\partial_H(x \lhd b \rhd y)$	$=$	$\partial_H(x) \lhd b \rhd \partial_H(y)$	
D6	$\partial_H(\sum_v p)$	$=$	$\sum_v \partial_H(p)$	
D7	$\partial_H(x\text{‹}t)$	$=$	$\partial_H(x)\text{‹}t$	
RD	$\rho_R(\delta)$	$=$	δ	
R1	$\rho_R(\mathsf{a}(\vec{d}))$	$=$	$\mathsf{a}(\vec{d})$	if $\mathsf{a} \notin \mathsf{dom}(R)$
R2	$\rho_R(\mathsf{a}(\vec{d}))$	$=$	$R(\mathsf{a})(\vec{d})$	if $\mathsf{a} \in \mathsf{dom}(R)$
R3	$\rho_R(x+y)$	$=$	$\rho_R(x)+\rho_R(y)$	
R4	$\rho_R(x\cdot y)$	$=$	$\rho_R(x)\cdot\rho_R(y)$	
R5	$\rho_R(x \lhd b \rhd y)$	$=$	$\rho_R(x) \lhd b \rhd \rho_R(y)$	
R6	$\rho_R(\sum_v p)$	$=$	$\sum_v \rho_R(p)$	
R7	$\rho_R(x\text{‹}t)$	$=$	$\rho_R(x)\text{‹}t$	

Table 2.10: Axioms of μCRL$_t$, where $x, y \in V_{\mathbb{P}}$, process-closed $p \in T_{\mathbb{P}}$, $\mathsf{a} \in$ *Act*, $b \in T_{\mathbb{B}}$, $t \in T_{\mathbb{T}}$, $v \in V$, $H \subseteq$ *Act* and R:*Act* $\rightarrow$ *Act*.

$$\frac{\mathsf{p} \xrightarrow{\mathsf{a}(\vec{d})}_t \surd \quad \mathsf{q} \xrightarrow{\mathsf{b}(\vec{d})}_t \surd \quad \gamma(\mathsf{a},\mathsf{b}) = \mathsf{c}}{\mathsf{p} \| \mathsf{q} \xrightarrow{\mathsf{c}(\vec{d})}_t \surd} \qquad \frac{\mathsf{p} \xrightarrow{\mathsf{a}(\vec{d})}_t \surd \quad \mathsf{q} \xrightarrow{\mathsf{b}(\vec{d})}_t \surd \quad \gamma(\mathsf{a},\mathsf{b}) = \mathsf{c}}{\mathsf{p} | \mathsf{q} \xrightarrow{\mathsf{c}(\vec{d})}_t \surd}$$

$$\frac{\mathsf{p} \xrightarrow{\mathsf{a}}_t \surd \quad \mathcal{U}_t(\mathsf{q})}{\mathsf{p} \ll \mathsf{q} \xrightarrow{\mathsf{a}}_t \surd} \qquad \frac{\mathsf{p} \xrightarrow{\mathsf{a}}_t \surd \quad \mathsf{a} \notin H}{\partial_H(\mathsf{p}) \xrightarrow{\mathsf{a}}_t \surd}$$

$$\frac{\mathsf{p} \xrightarrow{\mathsf{a}}_t \surd \quad \mathsf{a} \notin \mathsf{dom}(R)}{\rho_R(\mathsf{p}) \xrightarrow{\mathsf{a}}_t \surd} \qquad \frac{\mathsf{p} \xrightarrow{\mathsf{a}}_t \surd \quad \mathsf{a} \in \mathsf{dom}(R)}{\rho_R(\mathsf{p}) \xrightarrow{R(\mathsf{a})}_t \surd}$$

Table 2.11: Successful termination for μCRL$_t$, where $\mathsf{a} \in \mathcal{A}$, $\mathsf{a}, \mathsf{b}, \mathsf{c} \in$ *Act*, $\vec{d}, \vec{e} \in \mathcal{D}_s$, $\mathsf{p}, \mathsf{q} \in \mathcal{P}$, $H \subseteq$ *Act*, R:*Act* $\rightarrow$ *Act* and $t \in \mathcal{D}_{\mathbb{T}}$.

$$\frac{\mathsf{p} \xrightarrow{\mathsf{a}}_t \surd \quad \mathcal{U}_t(\mathsf{q})}{\mathsf{p}\|\mathsf{q} \xrightarrow{\mathsf{a}}_t t{\gg}\,\mathsf{q}} \qquad \frac{\mathsf{q} \xrightarrow{\mathsf{a}}_t \surd \quad \mathcal{U}_t(\mathsf{p})}{\mathsf{p}\|\mathsf{q} \xrightarrow{\mathsf{a}}_t t{\gg}\,\mathsf{p}} \qquad \frac{\mathsf{p} \xrightarrow{\mathsf{a}}_t \surd \quad \mathcal{U}_t(\mathsf{q})}{\mathsf{p} \mathbin{\lfloor\!\lfloor} \mathsf{q} \xrightarrow{\mathsf{a}}_t t{\gg}\,\mathsf{q}}$$

$$\frac{\mathsf{p} \xrightarrow{\mathsf{a}}_t \mathsf{p}' \quad \mathcal{U}_t(\mathsf{q})}{\mathsf{p}\|\mathsf{q} \xrightarrow{\mathsf{a}}_t \mathsf{p}'\|t{\gg}\,\mathsf{q}} \qquad \frac{\mathsf{q} \xrightarrow{\mathsf{a}}_t \mathsf{q}' \quad \mathcal{U}_t(\mathsf{p})}{\mathsf{p}\|\mathsf{q} \xrightarrow{\mathsf{a}}_t t{\gg}\,\mathsf{p}\|\mathsf{q}'} \qquad \frac{\mathsf{p} \xrightarrow{\mathsf{a}}_t \mathsf{p}' \quad \mathcal{U}_t(\mathsf{q})}{\mathsf{p} \mathbin{\lfloor\!\lfloor} \mathsf{q} \xrightarrow{\mathsf{a}}_t \mathsf{p}'\|t{\gg}\,\mathsf{q}}$$

$$\frac{\mathsf{p} \xrightarrow{\mathsf{a}(\vec{d})}_t \mathsf{p}' \quad \mathsf{a} \notin H}{\partial_H(\mathsf{p}) \xrightarrow{\mathsf{a}(\vec{d})}_t \partial_H(\mathsf{p}')} \qquad \frac{\mathsf{p} \xrightarrow{\mathsf{a}(\vec{d})}_t \mathsf{p}' \quad \mathsf{a} \notin \mathsf{dom}(R)}{\rho_R(\mathsf{p}) \xrightarrow{\mathsf{a}(\vec{d})}_t \rho_R(\mathsf{p}')} \qquad \frac{\mathsf{p} \xrightarrow{\mathsf{a}(\vec{d})}_t \mathsf{p}' \quad \mathsf{a} \in \mathsf{dom}(R)}{\rho_R(\mathsf{p}) \xrightarrow{R(\mathsf{a})(\vec{d})}_t \rho_R(\mathsf{p}')}$$

$$\frac{\mathsf{p} \xrightarrow{\mathsf{a}(\vec{d})}_t \surd \quad \mathsf{q} \xrightarrow{\mathsf{b}(\vec{d})}_t \mathsf{q}' \quad \gamma(\mathsf{a},\mathsf{b}) = \mathsf{c}}{\mathsf{p}\|\mathsf{q} \xrightarrow{\mathsf{c}(\vec{d})}_t \mathsf{q}'} \qquad \frac{\mathsf{p} \xrightarrow{\mathsf{a}(\vec{d})}_t \mathsf{p}' \quad \mathsf{q} \xrightarrow{\mathsf{b}(\vec{d})}_t \surd \quad \gamma(\mathsf{a},\mathsf{b}) = \mathsf{c}}{\mathsf{p}\|\mathsf{q} \xrightarrow{\mathsf{c}(\vec{d})}_t \mathsf{p}'}$$

$$\frac{\mathsf{p} \xrightarrow{\mathsf{a}(\vec{d})}_t \mathsf{p}' \quad \mathsf{q} \xrightarrow{\mathsf{b}(\vec{d})}_t \mathsf{q}' \quad \gamma(\mathsf{a},\mathsf{b}) = \mathsf{c}}{\mathsf{p}\|\mathsf{q} \xrightarrow{\mathsf{c}(\vec{d})}_t \mathsf{p}'\|\mathsf{q}'} \qquad \frac{\mathsf{p} \xrightarrow{\mathsf{a}(\vec{d})}_t \surd \quad \mathsf{q} \xrightarrow{\mathsf{b}(\vec{d})}_t \mathsf{q}' \quad \gamma(\mathsf{a},\mathsf{b}) = \mathsf{c}}{\mathsf{p}|\mathsf{q} \xrightarrow{\mathsf{c}(\vec{d})}_t \mathsf{q}'}$$

$$\frac{\mathsf{p} \xrightarrow{\mathsf{a}(\vec{d})}_t \mathsf{p}' \quad \mathsf{q} \xrightarrow{\mathsf{b}(\vec{d})}_t \surd \quad \gamma(\mathsf{a},\mathsf{b}) = \mathsf{c}}{\mathsf{p}|\mathsf{q} \xrightarrow{\mathsf{c}(\vec{d})}_t \mathsf{p}'} \qquad \frac{\mathsf{p} \xrightarrow{\mathsf{a}(\vec{d})}_t \mathsf{p}' \quad \mathsf{q} \xrightarrow{\mathsf{b}(\vec{d})}_t \mathsf{q}' \quad \gamma(\mathsf{a},\mathsf{b}) = \mathsf{c}}{\mathsf{p}|\mathsf{q} \xrightarrow{\mathsf{c}(\vec{d})}_t \mathsf{p}'\|\mathsf{q}'}$$

$$\frac{\mathsf{p} \xrightarrow{\mathsf{a}}_t \mathsf{p}' \quad \mathcal{U}_t(\mathsf{q})}{\mathsf{p} \ll \mathsf{q} \xrightarrow{\mathsf{a}}_t \mathsf{p}'}$$

Table 2.12: Transition relation for μCRL$_t$, where $\mathsf{a} \in \mathcal{A}$, $\mathsf{a}, \mathsf{b}, \mathsf{c} \in \mathit{Act}$, $\mathsf{p}, \mathsf{q}, \mathsf{p}', \mathsf{q}' \in \mathcal{P}$, $H \subseteq \mathit{Act}$, $R{:}\mathit{Act} \to \mathit{Act}$, $\vec{d}, \vec{e} \in \mathcal{D}_s$ and $t \in T_{\mathbb{T}}$.

$$\frac{\mathcal{U}_t(\mathsf{p}) \quad \mathcal{U}_t(\mathsf{q})}{\mathcal{U}_t(\mathsf{p}\|\mathsf{q})} \qquad \frac{\mathcal{U}_t(\mathsf{p}) \quad \mathcal{U}_t(\mathsf{q})}{\mathcal{U}_t(\mathsf{p} \mathbin{\lfloor\!\lfloor} \mathsf{q})} \qquad \frac{\mathcal{U}_t(\mathsf{p}) \quad \mathcal{U}_t(\mathsf{q})}{\mathcal{U}_t(\mathsf{p}|\mathsf{q})}$$

$$\frac{\mathcal{U}_t(\mathsf{p}) \quad \mathcal{U}_t(\mathsf{q})}{\mathcal{U}_t(\mathsf{p} \ll \mathsf{q})} \qquad \frac{\mathcal{U}_t(\mathsf{p})}{\mathcal{U}_t(\partial_H(\mathsf{p}))} \qquad \frac{\mathcal{U}_t(\mathsf{p})}{\mathcal{U}_t(\rho_R(\mathsf{p}))}$$

Table 2.13: Deduction rules for the delay relation for μCRL$_t$, where $\mathsf{p}, \mathsf{q} \in \mathcal{P}$, $H \subseteq \mathit{Act}$, $R{:}\mathit{Act} \to \mathit{Act}$ and $t \in T_{\mathbb{T}}$.

Definition 2.5.1 (*Linear Process Equation*).
A process equation is called a *Linear Process Equation* if it is of the form of Eqn. (2.26)

$$\begin{array}{rl} P(d{:}D) = & \sum_{i\in I} \sum_{e_i:D_i} [b_i(d,e_i)] ::\!\rightarrow a_i(f_i(d,e_i)) \cdot P(g_i(d,e_i)) + \\ & \sum_{j\in J} \sum_{e_j:D_j} [b_j(d,e_j)] ::\!\rightarrow a_j(f_j(d,e_j)) \end{array} \tag{2.26}$$

where, I, J are finite sets of indices, $a_n{:}D_{s_{\mathsf{dom}(f_n)}} \rightarrow \mathbb{P} \in \mathit{Act}$ is an action declaration, for $n \in I \cup J$, $f_n{:}D \times D_n \rightarrow D_{s_{f_n}}$ is a function yielding an action parameter for $n \in I \cup J$, $g_i{:}D \times D_i \rightarrow D$ is a function, yielding the next state and $b_n{:}D \times D_n \rightarrow \mathbb{B}$ is a function, yielding a guard for action $a_n(f_n(d,e_n))$, for $n \in I \cup J$.

In this thesis, we mostly restrict to representing non-successfully terminating processes. This means that for the index set J, all action declarations are equal to the constant δ. Using axiom A7, we can then merge the sets I and J in the definition of an LPE.

The timed version of an LPE is similar to the untimed version of an LPE, and is in this thesis referred to as the *Timed Linear Process Equation* (TLPE).

Definition 2.5.2 (*Timed Linear Process Equation*).
A process equation is called a *Timed Linear Process Equation* if it is of the form of Eqn. (2.27)

$$\begin{array}{rl} P(d{:}D) = & \sum_{i\in I} \sum_{e_i:D_i} [b_i(d,e_i)] ::\!\rightarrow a_i(f_i(d,e_i))\text{‹}(g_i(d,e_i)) \cdot P(h_i(d,e_i)) + \\ & \sum_{j\in J} \sum_{e_j:D_j} [b_j(d,e_j)] ::\!\rightarrow a_j(f_j(d,e_j))\text{‹}(g_j(d,e_j)) \end{array} \tag{2.27}$$

where, I and J are finite sets of indices, $a_n{:}D_{s_{\mathsf{dom}(f_n)}} \rightarrow \mathbb{P} \in \mathit{Act}$ is an action declaration, for $n \in I \cup J$, $f_n{:}D \times D_n \rightarrow D_{s_{f_n}}$ is a function yielding an action parameter for $n \in I \cup J$, $g_n{:}D \times D_n \rightarrow \mathbb{T}$ is a function, yielding a moment in time for $n \in I \cup J$, $h_i{:}D \times D_i \rightarrow D$ is a function, yielding the next state and $b_n{:}D \times D_n \rightarrow \mathbb{B}$ is a function, yielding a guard for action $a_n(f_n(d,e_n))$, for $n \in I \cup J$.

Bibliographic Notes

The untimed and timed process algebras we discussed in this chapter are the result of several years of research. Their definitions are influenced by the ideas that grew out of the concurrent development of many other process algebras. In particular, the language μCRL was heavily influenced by ACP and the ideas that came forth from the CRL project. In turn, the investigations into timed extensions of ACP largely influenced the definition of μCRL_t. For a recent exposition of the various possible extensions of ACP with timing, we refer to [17].

The definition of μCRL_t also benefitted a lot from the expertise that was obtained by investigations into timed extensions of other formalisms. Noteworthy is the international project that aimed at defining E-LOTOS [68]. In defining E-LOTOS – which only recently became an international standard – much effort was spent on improving on the standardised language LOTOS [24] by defining e.g. new semantical models for the data language and including predefined and external data types. More importantly, in this project, a great deal of effort was spent in obtaining a timed extension of LOTOS. To this end, several possibilities were studied, resulting in variants of standard LOTOS. Here, we name but a few. Léonard and Leduc defined ET-LOTOS [82, 82], Leduc defined TLOTOS in [81] and LOTOS NT [47, 113] was defined by Garavel and Sighireanu. The latter is based on an early version of E-LOTOS itself. The LOTOS NT variant was derived amongst other reasons because of the complexity of E-LOTOS (see also [47]). In ET-LOTOS it is, like in μCRL$_t$, possible to use an unspecified time domain. Timing constraints are given by means of an *extended action prefix operator*. In TLOTOS, a discrete time domain is used. Three timing constructs can be identified, viz. a *start delay operator*, an *execution delay operator* and an *unbounded start delay*. TLOTOS is closely related to ATP [99]. The latter is compared to ACP with discrete timing in [120].

Whereas μCRL$_t$ aims at being applied in practice and being used for conducting theoretical investigations such as expressiveness, completeness, etc. E-LOTOS is tailored, mainly to deal with the former aspect: several exercises have been conducted using E-LOTOS, or some of the above-mentioned versions of timed LOTOS, see e.g. [118, 114, 111]. For μCRL$_t$, only two case-studies [58, 123], but a larger number of theoretical exercises, see e.g. [57, 108, 119] were conducted.

Lastly, we mention several timed extensions of CCS [97] that have been developed. Chen [36] describes a timed process algebra, called Timed CCS, that uses an unspecified time domain. Timing is introduced via an *action prefix operator*. The Linear Timed CCS by Jeffrey [72] uses an unspecified, totally ordered monoid as its time domain. The author shows that CCS and many of its timed extensions can be mimicked by his process algebra. Moller and Tofts [98] introduce Temporal CCS, which is basically plain CCS to which several discrete timing constructs are added. Finally, Wang [121, 122] describes a timed extension of CCS, also called Timed CCS. It allows for an unspecified time domain and has two main constructs for specifying the passage of time.

The most apparent difference between the above-mentioned languages and μCRL$_t$ seems to be the fact that time is dealt with as part of the data language of μCRL$_t$. The upshot of this is that it is possible to translate μCRL$_t$ to plain μCRL (see [119, 108]), and thus, μCRL$_t$ is in a sense not very different from plain μCRL. We are not aware of any other process algebra for which similar results have been obtained; all other process algebras seem to really add constructs to the basic language that cannot be eliminated.

For a more detailed overview of the above-mentioned and other languages, such as timed extensions of CSP, we refer to [73, 120, 11].

Chapter 3

Verification of Data-Dependent Systems

Model checking has emerged as one of the most prominent techniques for automated reasoning about formal models. Techniques for model checking have made major advances in the last decade (see e.g. [33, 27]), and it has earned its medals in many application areas. The class of systems that are amenable for model checking, however, is generally restricted to systems in which data dependencies are absent or can be abstracted from. Notable exceptions are a few classes of real-time and hybrid systems [7, 6]. Unfortunately, most data-dependent systems are left without appropriate techniques for fully-automatic verification. This chapter bridges this gap by introducing a technique for model checking data-dependent systems.

In its most straightforward case, verifying systems using model checking boils down to finding a proof of satisfaction of a logical formula (representing a property) given a directed graph of states and action labelled edges (representing a system). Constructing an explicit representation of a system, however, is not always possible, and in general is quite impossible when dealing with unbounded data types. For such systems, *on the fly* generation of the underlying transition system of the system is more appropriate, since only the states and edges that matter are computed. Given the undecidability of the model checking problem of data-dependent systems in general, we cannot hope our proposed solution can deal with all systems. However, our experiments with a prototype implementation of our technique have shown the usefulness of model checking data-dependent systems for many interesting problems.

The investigations into the automated verification of data-dependent systems have a short history. A closely related topic is the automated verification of parameterised systems. Such systems actually represent a class of systems, differing only in the values of global parameters. Data-dependent systems can also depend on such parameters, however, on top of this, such systems also depend on internally communicated and used data values. The current state of the art in the verification of parameterised systems includes e.g. [22, 110, 42].

This chapter is structured as follows. In Section 3.1, we introduce the logic in which we express properties of a system. This logic is interpreted over a labelled transition system induced by a Linear Process Equation (see Chapter 2). Section 3.2 then describes fixpoint equation systems [88], tailored to dealing with data and quantifiers. This formalism is used as an intermediate formalism in which the system under verification and a logical property are expressed. Section 3.3 then provides an algorithm for computing a solution to these (extended) equation systems. In Section 3.4 we discuss a prototype implementation of this algorithm, and describe several results obtained by using this prototype implementation. Finally, Section 3.5 places the obtained results into perspective and discusses some related work.

3.1 First Order Modal μ-Calculus

The logic we consider is based upon the modal μ-calculus [75], extended with data variables, quantifiers and parameterisation (see [51]). This logic allows us to express data dependent properties. We refer to this logic as the *first order modal μ-calculus*. The logic is interpreted over a labelled transition system, induced by an LPE (see Def. 2.5.1). In this chapter, we focus on processes that do not terminate successfully. For this reason, we here formulate the definition of an LPE in a slightly different way.

Definition 3.1.1 (*Linear Process Equation*).
A process equation is called a *Linear Process Equation* if it is of the form of Eqn. (3.1)

$$P(d{:}D) = \sum_{i\in I} \sum_{e_i:D_i} [b_i(d,e_i)] ::\to a_i(f_i(d,e_i)) \cdot P(g_i(d,e_i)) \tag{3.1}$$

where, I is a finite set of indices, D and D_i are data domains; d and e_i are data variables and for $i \in I$, a_i are actions with parameters of sort D_{a_i}; $f_i{:}D \times D_i \to D_{a_i}$ $g_i{:}D \times D_i \to D$ and $b_i{:}D \times D_i \to \mathbb{B}$ are functions.

The operational semantics for μCRL can be found in Chapter 2. Since we restrict our discussion to process expressions in LPE-form, we here provide a definition of the labelled transition system as it is induced by a process in LPE-form

Definition 3.1.2 (*Transition System of an LPE*).
The *labelled transition system* of a Linear Process Equation as defined in Def. 3.1.1 is a quadruple $M = \langle S, \Sigma, \to, s_0\rangle$, where

- $S = \{X(d) \mid d{\in}D\}$ is the (possibly infinite) set of *states*;
- $\Sigma = \{a_i(d_{a_i}) \mid i \in I \wedge a_i \in \mathit{Act} \wedge d_{a_i} \in D_{a_i}\}$ is the (possibly infinite) set of labels;
- $\to = \{(X(d), a_i(d'_a), X(d')) \mid i \in I \wedge a_i \in \mathit{Act} \wedge \exists_{e_i\in D_i} c_i(d,e_i) \wedge d'_a = f_i(d,e_i) \wedge d' = g_i(d,e_i)\}$ is the *transition relation*. Rather than $(X(d), a(e), X(d')) \in\to$, we write $X(d) \xrightarrow{a(e)} X(d')$;
- $s_0 = X(d_0){\in}S$, for a given $d_0{\in}D$, is the *initial state*.

We proceed by defining the syntax and semantics of the first order modal μ-calculus.

Definition 3.1.3 (*Action Formulae*).
An *Action Formula* is defined by the grammar, defined by Eqn. (3.2).

$$\alpha ::= a(e) \mid \mathsf{t} \mid \mathsf{f} \mid \neg\alpha_1 \mid \alpha_1 \wedge \alpha_2 \mid \alpha_1 \vee \alpha_2 \mid \exists d{:}D.\alpha_1 \mid \forall d{:}D.\alpha_1 \tag{3.2}$$

Here, a is a parameterised action of set *Act*, d is a data variable of some set of data variables and e is some data expression of the datatype D.

The action formulae are interpreted over a labelled transition system $M = \langle S, \Sigma, \rightarrow, s_0 \rangle$, which is induced by an LPE (see Def. 3.1.1). The variables are interpreted in the context of an *environment*.

Definition 3.1.4 (*Environment*).
An environment is a mapping that assigns to each variable of a set of variables, a value of some given type.

Notation 3.1.5. Throughout this chapter, we use the following notational convention. For an environment η, we write $\eta[v/d]$ for the environment η', defined as $\eta'(d') = \eta(d')$ for all $d' \not\equiv d$ and $\eta'(d) = v$. In effect, $\eta[v/d]$ stands for the environment η where the value of d has changed to v. The interpretation of a variable d in an environment η is written as $\eta(d)$.

Action formulae are interpreted in the context of a data environment ε for data variables. For a given data variable d, $\varepsilon(d)$ yields a value of type D.

Definition 3.1.6 (*Interpretation of Action Formulae*).
Let ε be a data environment and α be an action formula. The interpretation of α in the context of data environment ε is denoted $[\![\alpha]\!]\varepsilon$, and is defined inductively in Eqn. (3.3).

$$\begin{array}{ll}
[\![\mathsf{t}]\!]\varepsilon & = \Sigma \\
[\![\mathsf{f}]\!]\varepsilon & = \emptyset \\
[\![a(e)]\!]\varepsilon & = \{a(\varepsilon(e))\} \\
[\![\neg\alpha]\!]\varepsilon & = \Sigma \setminus [\![\alpha]\!]\varepsilon \\
[\![\alpha_1 \wedge \alpha_2]\!]\varepsilon & = [\![\alpha_1]\!]\varepsilon \cap [\![\alpha_2]\!]\varepsilon \\
[\![\alpha_1 \vee \alpha_2]\!]\varepsilon & = [\![\alpha_1]\!]\varepsilon \cup [\![\alpha_2]\!]\varepsilon \\
[\![\exists d{:}D.\alpha]\!]\varepsilon & = \bigcup v{\in}D\ [\![\alpha]\!]\varepsilon[v/d] \\
[\![\forall d{:}D.\alpha]\!]\varepsilon & = \bigcap v{\in}D\ [\![\alpha]\!]\varepsilon[v/d]
\end{array} \tag{3.3}$$

Hence, we can use t to denote an arbitrary (parameterised) action. This is useful for expressing e.g. progress conditions. We subsequently define the set of *state formulae*. We present these in *Positive Normal Form*. This means that negation only occurs on the level of atomic propositions and, in addition, all bound variables are distinct.

Definition 3.1.7 (*State Formulae*).
A *State Formula* is given by the grammar, defined in Eqn. (3.4).

$$\begin{array}{ll}
\varphi & ::= b \mid Y(e) \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid [\alpha]\varphi_1 \mid \langle\alpha\rangle\varphi_1 \mid \\
& \quad \exists d{:}D.\varphi \mid \forall d{:}D.\varphi \mid (\mu Z(d{:}D).\varphi)(e) \mid (\nu Z(d{:}D).\varphi)(e)
\end{array} \tag{3.4}$$

where b is an expression of the domain $\mathbb{B}$, d is a data variable, e is some data expression, α is an action formula and Y is a propositional variable. We assume all names in $(\sigma X(d{:}D).\varphi)(e)$, where $\sigma{\in}\{\mu, \nu\}$ is a fixpoint operator, are unique, i.e. each variable is bound only once by a fixpoint operator or quantifier.

State formulae are again interpreted over a labelled transition system M, induced by an LPE, according to Def. 3.1.1. We interpret state formulae in the context of a propositional environment ρ for propositional variables and a data environment ε for data variables. For a given propositional variable X, $\rho(X)$ yields a mapping of type $D \rightarrow 2^S$.

Definition 3.1.8 (*Interpretation of State Formulae*).
Let ε be a data environment, ρ a propositional environment and let φ be a state formula. The interpretation of φ in the context of data environment ε and propositional environment ρ is denoted $[\![\varphi]\!]\rho\varepsilon$, and is defined inductively in Eqn. (3.5):

$$
\begin{array}{ll}
[\![b]\!]\rho\varepsilon & = \begin{cases} S \text{ if } [\![b]\!]\varepsilon \\ \emptyset \text{ otherwise} \end{cases} \\
[\![X(e)]\!]\rho\varepsilon & = (\rho(X))(\varepsilon(e)) \\
[\![\varphi_1 \wedge \varphi_2]\!]\rho\varepsilon & = [\![\varphi_1]\!]\rho\varepsilon \cap [\![\varphi_2]\!]\rho\varepsilon \\
[\![\varphi_1 \vee \varphi_2]\!]\rho\varepsilon & = [\![\varphi_1]\!]\rho\varepsilon \cup [\![\varphi_2]\!]\rho\varepsilon \\
[\![[\alpha]\varphi]\!]\rho\varepsilon & = \{X(v){\in}S \mid\ \forall v'{\in}D\ \forall a{\in}\mathit{Act}\ \forall v_a{\in}D_a \\
 & \qquad (X(v) \xrightarrow{a(v_a)} X(v') \wedge a(v_a){\in}[\![\alpha]\!]\varepsilon) \rightarrow X(v'){\in}[\![\varphi]\!]\rho\varepsilon\} \\
[\![\langle\alpha\rangle\varphi]\!]\rho\varepsilon & = \{X(v){\in}S \mid\ \exists v'{\in}D\ \exists a{\in}\mathit{Act}\ \exists v_a{\in}D_a \\
 & \qquad (X(v) \xrightarrow{a(v_a)} X(v') \wedge a(v_a){\in}[\![\alpha]\!]\varepsilon \wedge X(v'){\in}[\![\varphi]\!]\rho\varepsilon)\} \\
[\![\forall d{:}D.\varphi]\!]\rho\varepsilon & = \bigcap v'{\in}D\ [\![\varphi]\!]\rho(\varepsilon[v'/d]) \\
[\![\exists d{:}D.\varphi]\!]\rho\varepsilon & = \bigcup v'{\in}D\ [\![\varphi]\!]\rho(\varepsilon[v'/d]) \\
[\![(\mu Z(d{:}D).\varphi)(e)]\!]\rho\varepsilon & = (\bigcap\{X{:}D \rightarrow 2^S \mid [\![\varphi]\!](\rho[X/Z])\varepsilon \dot{\subseteq} X\})(\varepsilon(e)) \\
[\![(\nu Z(d{:}D).\varphi)(e)]\!]\rho\varepsilon & = (\bigcup\{X{:}D \rightarrow 2^S \mid X \dot{\subseteq} [\![\varphi]\!](\rho[X/Z])\varepsilon\})(\varepsilon(e))
\end{array}
\tag{3.5}
$$

Note: for $X{:}2^{D\rightarrow 2^S}$ and $d{\in}D$, we write $X(d)$ for the set of elements $\{x(d)|x{\in}X\}$.

Here, we define the ordering $\dot{\subseteq}$ on the set $D \rightarrow 2^S$ as $X\dot{\subseteq}Y$ iff for all $d{:}D$ we have $X(d) \subseteq Y(d)$. The set $(D \rightarrow 2^S, \dot{\subseteq})$ forms a complete lattice. From Lemma 3.1.9, stated below, the existence and uniqueness of fixpoints in state formulae immediately follows.

Lemma 3.1.9. The operator $\Psi{:}(D \rightarrow 2^S) \rightarrow (D \rightarrow 2^S)$ that is associated to state formula $(\sigma Z(d{:}D).\psi)(e)$, and defined as $\Psi = \lambda X{:}D \rightarrow 2^S.\lambda v{:}D.[\![\psi]\!](\rho[X/Z])(\varepsilon[v/d])$ for data environment ε and propositional environment ρ is monotonic over the complete lattice $(D \rightarrow 2^S, \dot{\subseteq})$.

Proof. Follows from the fact that the state formulae are presented in Positive Normal Form. □

The modal μ-calculus is quite expressive, but also renowned for its incomprehensibility. An enlightening explanation of the modal μ-calculus can be found in e.g. [29]. We here provide two examples, one of which uses the first order extensions of the modal μ-calculus, to illustrate typical formulae and we explain their meaning.

Example 3.1.10. An example of a modal μ-calculus formula is one that identifies processes for which progress is ensured. The expression $\nu X.([\mathsf{t}]X \wedge \langle \mathsf{t}\rangle \mathsf{t})$ expresses that we can "infinitely often" perform at least a single step. Thus, this expression must be interpreted as freedom of deadlock. Notice that every modal μ-calculus formula is indeed also a first order modal μ-calculus formula.

Example 3.1.11. Assume a process with at least the states s_0, s_1 and s_2, the labels $a(\mathsf{t})$ and $a(\mathsf{f})$ and the state formula φ (see Fig. 3.1). We write $s \models \varphi$ to denote that φ is satisfied in state s, and, likewise, we write $s \not\models \varphi$ to denote that φ is not satisfied in state s. We illustrate the difference in data-quantification in action formulae and data-quantification in state formulae. Then, we can formulate two properties, showing the distinction between data-quantification in action formulae and in state formulae:

1. The state formula $(\exists b{:}\mathbb{B}.\ [a(b)]\varphi)$ holds in state s_0. Basically, this expression states there exists a data-parameter b, such that after executing an action $a(b)$, we end up in a state satisfying φ.

2. The state formula $([\exists b{:}\mathbb{B}.a(b)]\varphi)$ does not hold in state s_0. This expression states that, whatever the value of the parameter of the action a is, we end up in a state satisfying φ, which, obviously, is not true.

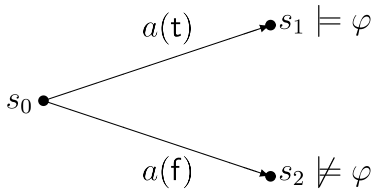


Figure 3.1: Example of a simple transition system.

Notice that data-quantification in action formulae can be used for abstracting from the actual values for parameterised actions.

There are several identities between action formulae and state formulae (see Lemma 3.1.12 below). Using these identities, we can rephrase the second state formula of the last example to the equivalent state formula $\forall b{:}\mathbb{B}.[a(b)]\varphi$, which makes the difference between both state formulae in that particular example more obvious.

Lemma 3.1.12. Let φ be a state formula, such that $d \notin FV(\varphi)$, and let α be an action formula. Then, we have the following identities:

- $\langle \exists d{:}D.\alpha \rangle \varphi \Leftrightarrow \exists d{:}D.\langle \alpha \rangle \varphi$,
- $[\exists d{:}D.\alpha]\varphi \Leftrightarrow \forall d{:}D.[\alpha]\varphi$,
- $\exists d{:}D.[\alpha]\varphi \Rightarrow [\forall d{:}D.\alpha]\varphi$,
- $\langle \forall d{:}D.\alpha \rangle \varphi \Rightarrow \forall d{:}D.\langle \alpha \rangle \varphi$

Note: here we use implication as an abbreviation for $\subseteq$ and bi-implication as an abbreviation for $=$ on the interpretations of the state formulae.

Proof. Follows immediately from the definition of the interpretations of action formulae and state formulae. □

Notice that the converse of the latter two identities is in general not true. For this, we consider the following example.

Example 3.1.13. Assume again a process with at least the states s_0, s_1 and s_2, the labels $a(\mathsf{t})$ and $a(\mathsf{f})$ and the state formula φ. Consider the part of this process visualised by Fig. 3.2. We show that the converse of the latter two identities in Lemma 3.1.12 does not hold.

1. The state formula $\forall b{:}\mathbb{B}.\langle a(b)\rangle\varphi$ obviously holds in state s_0: since the universal quantifier ranges over a finite domain, we can write this formula as $\langle a(\mathsf{t})\rangle\varphi \wedge \langle a(\mathsf{f})\rangle\varphi$. However, the state formula $\langle \forall b{:}\mathbb{B}.a(b)\rangle\varphi$ does not hold in state s_0: we can write this formula as $\langle \mathsf{f}\rangle\varphi$, which actually holds in no state.

2. Similarly, we can prove that state formula $[\forall b{:}\mathbb{B}.a(b)]\neg\varphi$ holds in state s_0. However, state formula $\exists b{:}\mathbb{B}.[a(b)]\neg\varphi$ does not hold in state s_0, since both transition $a(\mathsf{t})$ and $a(\mathsf{f})$ lead to a state where φ holds, contradicting the requirement that φ should not hold.

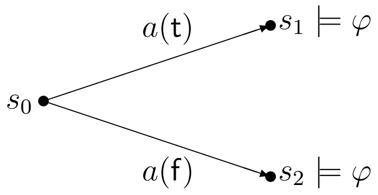


Figure 3.2: Example of another transition system.

This last example shows that the quantifiers inside action formulae in general cannot be removed in favour of the quantifiers in state formulae. Thus, if we compare the fragment of the first order modal μ-calculus that disallows the use of quantifiers inside action formulae with the whole first order modal μ-calculus, we observe that the quantifiers inside action formulae indeed add to the expressivity of the whole first order modal μ-calculus.

3.2 Equation Systems

We aim at verifying first order modal μ-calculus expressions on processes, specified as Linear Process Equations. For this, we follow the approach, outlined in e.g. [88]. In essence, we use an extension of the formalism of *boolean equation systems* as an intermediate formalism that allows us to combine a Linear Process Equation with a first order modal μ-calculus expression. Verification of the process then occurs on the level of these equation systems.

Definition 3.2.1 (*First Order Boolean Expression*).
A *first order boolean expression* is a formula φ in positive form, defined by the grammar given in Eqn. (3.6).

$$\varphi ::= b \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid X(e) \mid \forall d{:}D.\varphi \mid \exists d{:}D.\varphi \tag{3.6}$$

where b is an expression of datatype $\mathbb{B}$, X is a propositional variable of a set $\mathcal{X}$ of propositional variables, d is a data variable and e is a term of data-type D.

We define the ordering $\dot{\Rightarrow}$ on the set $D \to \mathbb{B}$ as $\varphi\dot{\Rightarrow}\psi$ iff for all $d{:}D$, we have $\varphi(d) \Rightarrow \psi(d)$. The set of first order boolean expressions $(D \to \mathbb{B}, \dot{\Rightarrow})$ forms a complete lattice and is in fact isomorphic to the powerset of the set D, where set inclusion is used as the ordering.

The propositional variables $X{\in}\mathcal{X}$, occurring as free variables in first order boolean expressions are bound in *first order boolean equation systems*, used in the sequel. The interpretation of the variables X is given by an environment θ for propositional variables, assigning functions of type $D \to \mathbb{B}$ to the variables in the set $\mathcal{X}$. We use the notational conventions (defined on page 33) for environments.

We define the ordering $\leq$ on the set $\mathcal{X} \to D \to \mathbb{B}$ as $\theta_1 \leq \theta_2$ iff for all $X{\in}\mathcal{X}$, we have $\theta_1(X) \dot{\Rightarrow} \theta_2(X)$. The set $(\mathcal{X} \to D \to \mathbb{B}, \leq)$ is (for fixed sets $\mathcal{X}$ and D), a complete lattice.

Definition 3.2.2 (*Interpretation of First Order Boolean Expression*).
Let θ be a propositional environment and η be a data environment. The *interpretation* of a first order boolean expression φ in the context of environments θ and η, written as $[\![\varphi]\!]\theta\eta$ is either true or false, determined by the induction, given in Eqn. (3.7).

$$\begin{array}{lcl}
[\![b]\!]\theta\eta & = & [\![b]\!]\eta \\
[\![\varphi_1 \wedge \varphi_2]\!]\theta\eta & = & [\![\varphi_1]\!]\theta\eta \wedge [\![\varphi_2]\!]\theta\eta \\
[\![\varphi_1 \vee \varphi_2]\!]\theta\eta & = & [\![\varphi_1]\!]\theta\eta \vee [\![\varphi_2]\!]\theta\eta \\
[\![X(e)]\!]\theta\eta & = & \theta(X)([\![e]\!]\eta) \\
[\![\forall d{:}D.\varphi]\!]\theta\eta & = & \begin{cases} \text{true, if for all } v{:}D \text{ it holds that } [\![\varphi]\!]\theta(\eta[v/d]) \\ \text{false, otherwise} \end{cases} \\
[\![\exists d{:}D.\varphi]\!]\theta\eta & = & \begin{cases} \text{true, if there exists an } v{:}D \text{ such that } [\![\varphi]\!]\theta(\eta[v/d]) \\ \text{false, otherwise} \end{cases}
\end{array} \tag{3.7}$$

Definition 3.2.3 (*First Order Boolean Equation System*).
A *first order boolean equation system* $\mathcal{E}$ is a finite sequence of equations of the form $\sigma X(d{:}D) = \varphi$. Here, σ represents either the greatest or least fixed points ν or μ, and $\varphi{:}D \to \mathbb{B}$ is a first order boolean expression. We use the convention of denoting the empty sequence of equations by ϵ, and we require that all bound variables in a first order boolean equation system are distinct.

In the sequel, we refer to first order boolean equation systems as *equation systems*. The equation system $\mathcal{E}'$ that is obtained by applying an environment θ to an equation system $\mathcal{E}$ is the equation system in which every free variable $X{\in}\mathcal{X}$ is assigned the value $\theta(X)$.

Definition 3.2.4 (*Solution to an Equation System*).
Given a propositional environment θ, and an equation system $\mathcal{E}$. The solution $\mathcal{E}\theta$ to the equation system $\mathcal{E}$ is an environment that is defined inductively by Eqn. (3.8) (see also e.g. [88], Definition 3.3), where σ is either the greatest fixpoint or the least fixpoint ν or μ.

$$\begin{array}{lcl}
[\epsilon]\theta & = & \theta \\
[(\sigma X(d{:}D) = \varphi)\mathcal{E}]\theta & = & [\mathcal{E}](\theta[\sigma X.\varphi([\mathcal{E}]\theta)/X])
\end{array} \tag{3.8}$$

where

$$\begin{array}{lcl}
\mu X.\varphi([\mathcal{E}]\theta) & = & \bigwedge\{\psi{:}D \to \mathbb{B} \mid \varphi([\mathcal{E}]\theta[\psi/X]) \dot{\Rightarrow} \psi\} \\
\nu X.\varphi([\mathcal{E}]\theta) & = & \bigvee\{\psi{:}D \to \mathbb{B} \mid \psi \dot{\Rightarrow} \varphi([\mathcal{E}]\theta[\psi/X])\}
\end{array}$$

The operators $\bigwedge$ and $\bigvee$ resp. denote the *greatest lower bound* and the *least upper bound* of the complete lattice $(D \to \mathbb{B}, \dot{\Rightarrow})$.

Lemma 3.2.5 (*Monotonicity of First Order Boolean Expressions*).
Let $X(d{:}D) = \varphi$ be an equation, let θ be a propositional environment and η a data environment. For an equation $X(d{:}D) = \varphi$, we define an operator $\Phi{:}(D \to \mathbb{B}) \to (D \to \mathbb{B})$ as $\Phi = \lambda F{:}D \to \mathbb{B}.\lambda v{:}D.[\![\varphi]\!](\theta[F/X])(\eta[v/d])$. The operator Φ is monotonic over the complete lattice $(D \to \mathbb{B}, \dot{\Rightarrow})$.

Proof. Assume we are given an equation $X(d{:}D) = \varphi$, a propositional environment θ and a data environment η, and assume we have first order boolean expressions $\psi_1, \psi_2{:}D \to \mathbb{B}$, for which we require $\psi_1 \dot{\Rightarrow} \psi_2$. We proceed by induction on the structure of φ.

- Suppose $\varphi \equiv b$. Then, $\Phi(\psi_1)$ equals $\lambda v{:}D.[\![b]\!](\theta[\psi_1/X])(\eta[v/d])$. As there is no occurrence of X in b, this is equivalent to $\lambda v{:}D.[\![b]\!](\eta[v/d])$. Using the same steps in reverse order, we find this is equivalent to $\lambda v{:}D.[\![b]\!](\theta[\psi_2/X])(\eta[v/d])$ and therefore $\Phi(\psi_2)$.

- Suppose $\varphi \equiv Y(e)$. Then, $\Phi(\psi_1)$ is equivalent to $\lambda v{:}D.\psi_1([\![e]\!])(\eta[v/d])^{(*)}$, given that $Y \equiv X$ (if not, then we are done immediately). Since $\psi_1 \dot{\Rightarrow} \psi_2$, we therefore also have that $^{(*)}$ is at most $\lambda v{:}D.\psi_2([\![e]\!])(\eta[v/d])$, which is equivalent to $\Phi(\psi_2)$.

- Suppose $\varphi \equiv \varphi_1 \wedge \varphi_2$. Assume for first order boolean expressions φ_1 and φ_2, we already have $\Phi_1(\psi_1) \dot{\Rightarrow} \Phi_1(\psi_1)$ and $\Phi_2(\psi_1) \dot{\Rightarrow} \Phi_2(\psi_2)$. Then, $\Phi(\psi_1)$ is equal to the conjunction of the functionals $\lambda v{:}D.[\![\varphi_1]\!](\theta[\psi_1/X])(\eta[v/d])$ and $\lambda v{:}D.[\![\varphi_2]\!](\theta[\psi_1/X])(\eta[v/d])$. By induction, we know this is at most the conjunction of $\lambda v{:}D.[\![\varphi_1]\!](\theta[\psi_2/X])(\eta[v/d])$ and $\lambda v{:}D.[\![\varphi_2]\!](\theta[\psi_2/X])(\eta[v/d])$, from which we can deduce $\Phi(\psi_2)$. Similarly, we prove $\Phi(\psi_1) \dot{\Rightarrow} \Phi(\psi_2)$ in the case of $\varphi \equiv \varphi_1 \vee \varphi_2$.

- Suppose $\varphi \equiv \forall e{:}D.\varphi_e$. Assume for first order boolean expressions φ_e, we already have $\Phi_e(\psi_1) \dot{\Rightarrow} \Phi_e(\psi_2)$ for all $e{:}D$. Then, $\Phi(\psi_1)$ is equal to $\lambda v{:}D.[\![\varphi_e]\!](\theta[\psi_1/X])(\eta[v/d][x/e])$ for all $x{:}D$. By induction, this is at most $\lambda v{:}D.[\![\varphi_e]\!](\theta[\psi_2/X])(\eta[v/d][x/e])$ for all $x{:}D$, which is equal to $\Phi(\psi_2)$. Similarly, we prove $\Phi(\psi_1) \dot{\Rightarrow} \Phi(\psi_2)$ in the case of $\varphi \equiv \exists e{:}D.\varphi_e$.

□

Lemma 3.2.6. Let θ, θ' be propositional environments and let $\mathcal{E}$ be an equation system. Then, if $\theta \leq \theta'$ we also have $[\mathcal{E}]\theta \leq [\mathcal{E}]\theta'$.

Proof. We use induction on the length of the equation system. Let $\theta \leq \theta'$ be propositional environments.

- suppose $\mathcal{E} = \epsilon$. Then, $[\epsilon]\theta = [\epsilon]\theta'$.

- Suppose $\mathcal{E}$ is of the form $(\sigma X(d{:}D) = \varphi)\mathcal{E}'$. Assume we have $[\mathcal{E}']\theta \leq [\mathcal{E}']\theta'$.

  Now, $[(\sigma X(d{:}D) = \varphi)\mathcal{E}']\theta \leq [(\sigma X(d{:}D) = \varphi)\mathcal{E}']\theta'$ follows from

  $[\mathcal{E}'](\theta[\sigma X(d{:}D).\varphi([\mathcal{E}']\theta)/X]) \leq [\mathcal{E}'](\theta'[\sigma X(d{:}D).\varphi([\mathcal{E}']\theta')/X])$ (see Def. 3.2.4).

  By induction, this holds, as $\theta[\sigma X(d{:}D).\varphi([\mathcal{E}']\theta)/X] \leq \theta'[\sigma X(d{:}D).\varphi([\mathcal{E}']\theta')/X]$, follows from $\theta \leq \theta'$ and Lemma 3.2.5.

□

We next discuss how to use the formalism of equation systems as an intermediate formalism for solving the model-checking problem for processes with data. We define a translation that takes a Linear Process Equation and a first order modal μ-calculus formula and yields an equation system. Then, verifying a first order modal μ-calculus formula on an LPE is equivalent to calculating the solution to the equation system that takes the LPE and the expression as its input.

Definition 3.2.7. Let φ be a first order μ-calculus expression, such that φ is of the form $(\sigma X(d{:}D).\Phi)(e)$, and let $Y(d_p{:}D_p) = \sum_{i:I} \sum_{e_i:D_i} [c_i(d, e_i)] ::\!\rightarrow a_i(f_i(d, e_i)) \cdot Y(g_i(d, e_i))$ be a Linear Process Equation, where $d_p{:}D_p$ is the parameter of the process Y and a_i for $i{:}I$ is an action.
The equation system $\mathcal{E}$ that corresponds to the expression φ for the LPE Y, is given by $\mathbf{E}_{[]}(\varphi)$, where $[]$ denotes the empty list of parameters. The translation function $\mathbf{E}_{\vec{d_l}:\vec{D_l}}$ is defined by structural induction in Table 3.1, and action satisfaction, denoted by $\alpha \models \alpha'$ is defined using structural induction in Table 3.2 and assumes the process Y as given.

$$
\begin{array}{lcl}
\mathbf{E}_{\vec{d_l}:\vec{D_l}}(b) & \stackrel{\text{def}}{=} & \epsilon \\
\mathbf{E}_{\vec{d_l}:\vec{D_l}}(X(d_f{:}D_f)) & \stackrel{\text{def}}{=} & \epsilon \\
\mathbf{E}_{\vec{d_l}:\vec{D_l}}(\Phi_1 \wedge \Phi_2) & \stackrel{\text{def}}{=} & \mathbf{E}_{\vec{d_l}:\vec{D_l}}(\Phi_1)\mathbf{E}_{\vec{d_l}:\vec{D_l}}(\Phi_2) \\
\mathbf{E}_{\vec{d_l}:\vec{D_l}}(\Phi_1 \vee \Phi_2) & \stackrel{\text{def}}{=} & \mathbf{E}_{\vec{d_l}:\vec{D_l}}(\Phi_1)\mathbf{E}_{\vec{d_l}:\vec{D_l}}(\Phi_2) \\
\mathbf{E}_{\vec{d_l}:\vec{D_l}}([\alpha]\Phi) & \stackrel{\text{def}}{=} & \mathbf{E}_{\vec{d_l}:\vec{D_l}}(\Phi) \\
\mathbf{E}_{\vec{d_l}:\vec{D_l}}(\langle\alpha\rangle\Phi) & \stackrel{\text{def}}{=} & \mathbf{E}_{\vec{d_l}:\vec{D_l}}(\Phi) \\
\mathbf{E}_{\vec{d_l}:\vec{D_l}}(\forall d{:}D.\Phi) & \stackrel{\text{def}}{=} & \mathbf{E}_{\vec{d_l}:\vec{D_l};d:D}(\Phi) \\
\mathbf{E}_{\vec{d_l}:\vec{D_l}}(\exists d{:}D.\Phi) & \stackrel{\text{def}}{=} & \mathbf{E}_{\vec{d_l}:\vec{D_l};d:D}(\Phi) \\
\mathbf{E}_{\vec{d_l}:\vec{D_l}}((\sigma X(d_f{:}D_f).\Phi)(d)) & \stackrel{\text{def}}{=} & (\sigma \tilde{X}(d_f{:}D_f, d_p{:}D_p, \vec{d_l}{:}\vec{D_l}) = \tilde{\mathbf{E}}_{\vec{d_l}:\vec{D_l}}(\Phi)\ )\ \mathbf{E}_{\vec{d_l}:\vec{D_l}}(\Phi) \\
 & & \\
\tilde{\mathbf{E}}_{\vec{d_l}:\vec{D_l}}(b) & \stackrel{\text{def}}{=} & b \\
\tilde{\mathbf{E}}_{\vec{d_l}:\vec{D_l}}(X(d)) & \stackrel{\text{def}}{=} & \tilde{X}(d, d_p, \vec{d_l}) \\
\tilde{\mathbf{E}}_{\vec{d_l}:\vec{D_l}}(\Phi_1 \wedge \Phi_2) & \stackrel{\text{def}}{=} & \tilde{\mathbf{E}}_{\vec{d_l}:\vec{D_l}}(\Phi_1) \wedge \tilde{\mathbf{E}}_{\vec{d_l}:\vec{D_l}}(\Phi_2) \\
\tilde{\mathbf{E}}_{\vec{d_l}:\vec{D_l}}(\Phi_1 \vee \Phi_2) & \stackrel{\text{def}}{=} & \tilde{\mathbf{E}}_{\vec{d_l}:\vec{D_l}}(\Phi_1) \vee \tilde{\mathbf{E}}_{\vec{d_l}:\vec{D_l}}(\Phi_2) \\
\tilde{\mathbf{E}}_{\vec{d_l}:\vec{D_l}}([\alpha]\Phi) & \stackrel{\text{def}}{=} & \bigwedge_{i:I} \forall_{e_i:D_i}(a_i(f_i(d, e_i)) \models \alpha \wedge c_i(d, e_i)) \rightarrow \\
 & & \tilde{\mathbf{E}}_{\vec{d_l}:\vec{D_l}}(\Phi)[g_i(d, e_i)/d_p] \\
\tilde{\mathbf{E}}_{\vec{d_l}:\vec{D_l}}(\langle\alpha\rangle\Phi) & \stackrel{\text{def}}{=} & \bigvee_{i:I} \exists_{e_i:D_i}(a_i(f_i(d, e_i)) \models \alpha \wedge c_i(d, e_i) \wedge \\
 & & \tilde{\mathbf{E}}_{\vec{d_l}:\vec{D_l}}(\Phi)[g_i(d, e_i)/d_p]) \\
\tilde{\mathbf{E}}_{\vec{d_l}:\vec{D_l}}(\forall d{:}D.\Phi) & \stackrel{\text{def}}{=} & \forall d{:}D.\tilde{\mathbf{E}}_{\vec{d_l}:\vec{D_l};d:D}(\Phi) \\
\tilde{\mathbf{E}}_{\vec{d_l}:\vec{D_l}}(\exists d{:}D.\Phi) & \stackrel{\text{def}}{=} & \exists d{:}D.\tilde{\mathbf{E}}_{\vec{d_l}:\vec{D_l};d:D}(\Phi) \\
\tilde{\mathbf{E}}_{\vec{d_l}:\vec{D_l}}((\sigma X(d_f{:}D_f).\Phi)(d)) & \stackrel{\text{def}}{=} & \tilde{X}(d, d_p, \vec{d_l})
\end{array}
$$

Table 3.1: Translation of first order modal μ-calculus formula and LPE to an equation system. The propositional variable $\tilde{X}$ is a fresh variable, associated to a propositional variable X occurring in a first order modal μ-calculus formula.

The translation function $\mathbf{E}$ breaks down the μ-calculus expression given as an argument into several equations. The left-hand side of each equation is defined by the function $\mathbf{E}$, whereas its right-hand side is given by the function $\tilde{\mathbf{E}}$. Below, we illustrate the translation by means of a small example.

$$
\begin{array}{lcl}
a(d) \models a'(d') & \overset{\text{def}}{=} & a = a' \land d = d' \\
a(d) \models \mathsf{t} & \overset{\text{def}}{=} & \text{true} \\
a(d) \models \neg\alpha & \overset{\text{def}}{=} & \neg(a(d) \models \alpha) \\
a(d) \models \alpha_1 \land \alpha_2 & \overset{\text{def}}{=} & (a(d) \models \alpha_1) \land (a(d) \models \alpha_2) \\
a(d) \models \alpha_1 \lor \alpha_2 & \overset{\text{def}}{=} & (a(d) \models \alpha_1) \lor (a(d) \models \alpha_2) \\
a(d) \models \exists d'{:}D.\alpha & \overset{\text{def}}{=} & \exists d'{:}D.(a(d) \models \alpha) \\
a(d) \models \forall d'{:}D.\alpha & \overset{\text{def}}{=} & \forall d'{:}D.(a(d) \models \alpha)
\end{array}
$$

Table 3.2: Action Satisfaction

Example 3.2.8. Consider a coffee-vending machine that produces either cappuccino or espresso on the insertion of a special coin. The coffee-vending machine is clever enough to notice when it can no longer dispense a type of coffee; it accepts coins as long as there is at least one type of coffee that can still be dispensed. If the machine has run out of a type of coffee, it signals this type must be replaced (which is assumed to be done immediately after the signal).

$$
\begin{array}{rcl}
\textbf{proc } M(b{:}\mathbb{B}, c, e{:}\mathbb{N}) & = & [b \land c > 0] ::\rightarrow \textit{cappuccino} \cdot M(\neg b, c-1, e) \\
& + & [b \land e > 0] ::\rightarrow \textit{espresso} \cdot M(\neg b, c, e-1) \\
& + & [\neg b \land c + e > 0] ::\rightarrow \textit{coin} \cdot M(\neg b, c, e) \\
& + & [\neg b \land c = 0] ::\rightarrow \textit{refill}_{\textit{cappuccino}} \cdot M(b, C, e) \\
& + & [\neg b \land e = 0] ::\rightarrow \textit{refill}_{\textit{espresso}} \cdot M(b, c, E)
\end{array}
\tag{3.9}
$$

Here, the boolean b indicates whether a coin has been inserted or not; the variable c, resp. e registers the number of servings of cappuccino, reps. espresso are left in the coffee-vending machine. Now, consider the first order modal μ-calculus expression that expresses that if cappuccino is the only beverage that is ordered, then eventually, cappuccino is refilled, i.e. $\mu Z.[\textit{coin} \lor \textit{cappuccino}]Z \lor \langle \textit{refill}_{\textit{cappuccino}} \rangle \mathsf{t}$, Following the translation of Def. 3.2.7, we obtain the equation system (consisting of a single equation only) given in Eqn. (3.10).

$$
\begin{array}{rl}
\mu\tilde{Z}(b{:}\mathbb{B}, c, e{:}\mathbb{N}) = & ((\neg b \land c + e > 0) \rightarrow \tilde{Z}(\neg b, c, e)) \lor \\
& ((b \land c > 0) \rightarrow \tilde{Z}(\neg b, c-1, e)) \lor \\
& (\neg b \land c = 0)
\end{array}
\tag{3.10}
$$

Notice that, even though the first order modal μ-calculus expression did not use parameterised variables, the resulting equation system consists of an equation carrying the parameters of the Linear Process Equation.

We continue by establishing two results that allow us to define an algorithm for computing a solution to an equation system. The first lemma states that for an arbitrary equation system, we may replace an occurrence of an equation variable with its first order boolean expression in all equations prior to its defining equation. Note that, since all bound variables are distinct, there is in fact at most one defining equation for each variable.

Lemma 3.2.9. Let $\mathcal{E}_1, \mathcal{E}_2$ and $\mathcal{E}_3$ be equation systems and let $\sigma_1 X_1(d{:}D) = \varphi$ and $\sigma_2 X_2(e{:}D) = \psi$ be equations. Then, the identity of Eqn. (3.11) holds.

$$\begin{array}{ll} & [\![\mathcal{E}_1(\sigma_1 X_1(d{:}D) = \varphi)\mathcal{E}_2(\sigma_2 X_2(e{:}D) = \psi)\mathcal{E}_3]\!]\theta \\ = & \\ & [\![\mathcal{E}_1(\sigma_1 X_1(d{:}D) = \varphi[\psi/X_2])\mathcal{E}_2(\sigma_2 X_2(e{:}D) = \psi)\mathcal{E}_3]\!]\theta \end{array} \tag{3.11}$$

Proof. The proof is analogous to the proof of Lemma 6.3 in [88]. □

If we have the solution to a single equation in an equation system, then we can remove this equation from the equation system and update the environment to store the solution to this single equation. This means that if we can successively solve all single equations, the solution of the entire equation system follows.

Lemma 3.2.10. Let $\mathcal{E}, \mathcal{E}'$ be equation systems and let $\sigma X(d{:}D) = \psi$ be an equation, where $X \notin FV(\psi)$. Let θ be an arbitrary propositional environment. Then $[\mathcal{E}(\sigma X(d{:}D) = \psi)\mathcal{E}']\theta = [\mathcal{E}\mathcal{E}']\theta[\psi/X]$.

Proof. The proof proceeds by induction on the size of the equation system $\mathcal{E}$.

- $[(\sigma X(d{:}D) = \psi)\mathcal{E}']\theta$ is by definition equivalent to $[\mathcal{E}'](\theta[\sigma X(d{:}D).\psi/X])$, which in turn is equivalent to $[\mathcal{E}'](\theta[\psi/X])$, as since $X \notin FV(\psi)$, $\sigma X(d{:}D).\psi$ is ψ.
- Suppose $\mathcal{E}$ is of the form $(\sigma' Y(d{:}D) = \varphi)\mathcal{E}_0$, and assume $[\mathcal{E}_0(\sigma X(d{:}D) = \psi)\mathcal{E}']\theta = [\mathcal{E}_0\mathcal{E}']\theta[\psi/X]$ for all environments θ. Then, $[\mathcal{E}(\sigma X(d{:}D) = \psi)\mathcal{E}']\theta$ is by definition equivalent to $[\mathcal{E}_0(\sigma X(d{:}D) = \psi)\mathcal{E}'](\theta[\sigma' Y(d{:}D).\varphi/Y])$, which is by induction equivalent to $[\mathcal{E}_0\mathcal{E}'](\theta[\sigma' Y(d{:}D).\varphi/Y])[\psi/X])$. Again, by definition, this is equivalent to $[\mathcal{E}\mathcal{E}']\theta[\psi/X]$.

□

Due to Lemma 3.2.5 and the fact that the lattice $(D \to \mathbb{B}, \dot{\Rightarrow})$ is also a complete lattice, we know, by Tarski's theorem [116], that the least and greatest fixed points exist. These fixed points can be found by means of *approximation*.

Definition 3.2.11 (*Approximant Terms*).
Let $(D \to \mathbb{B}, \dot{\Rightarrow})$ be our complete lattice of first order boolean expressions, and let $\Phi{:}(D \to \mathbb{B}) \to (D \to \mathbb{B})$ be the operator that is associated to a first order boolean expression. Then $\sigma^\alpha X.\Phi$ is an approximant term, where α is an ordinal. We define the approximant terms by means of transfinite induction in Eqn. (3.12).

$$\begin{array}{lcl} \mu^0 X.\Phi(X) & = & \mathsf{f} \\ \nu^0 X.\Phi(X) & = & \mathsf{t} \\ \sigma^{\alpha+1} X.\Phi(X) & = & \Phi(\sigma^\alpha X.\Phi(X)) \\ \mu^\lambda X.\Phi(X) & = & \bigvee_{\alpha<\lambda} \mu^\alpha X.\Phi(X) \\ \nu^\lambda X.\Phi(X) & = & \bigwedge_{\alpha<\lambda} \nu^\alpha X.\Phi(X) \end{array} \tag{3.12}$$

3.3 Algorithm

Mader [88] describes an algorithm for solving boolean equation systems. The method she uses resembles the well-known Gauß elimination algorithm for solving linear equation systems, and is therefore also referred to as Gauß elimination. The algorithm we use (see Table 3.3) is an extension of the Gauß elimination algorithm of [88]. The essential difference is the addition of an extra loop for calculating a stable point in the approximation for each first order boolean equation.

The reduction of a first order boolean equation system proceeds in two separate steps. First, a stabilisation step is issued, in which a first order boolean equation $\sigma_i X_i(d{:}D) = \varphi_i$ is reduced to a stable equation $\sigma_i X_i(d{:}D) = \varphi'_i$, where φ'_i is an expression containing no occurrences of X_i. Second, we substitute each occurrence of X_i by φ'_i in the rest of the equations of the first order boolean equation system. Since there are no more occurrences of X_i in the right-hand side of the equations, it suffices to reduce a smaller first order boolean equation system. The algorithm terminates iff the approximation step terminates for each first order boolean equation.

Input: $(\sigma_1 X_1(d_1{:}D_1) = \varphi_1) \ldots (\sigma_n X_n(d_n{:}D_n) = \varphi_n)$.

```
1.   $i := n;$
2.   **while not** $i = 0$
3.     **do**
4.       $j := 0; \psi_0 := \sigma_{b_i};$
5.       **repeat**
6.           $\psi_{j+1} := \varphi_i[X_i := \psi_j];$
7.           $j := j + 1$
8.       **until** $(\psi_j \equiv \psi_{j-1})$
9.       $\varphi_i := \psi_j;$
10.      **for** $k = 1$ **to** $i - 1$ **do** $\varphi_k := \varphi_k[X_i := \varphi_i]$ **od** ;
11.      $i := i - 1$
12.    **od**
```

Remark: σ_{b_i} is t if $\sigma_i = \nu$, else f

Table 3.3: Algorithm for computing the solution of an equation system

Theorem 3.3.1 (*Soundness*).
On termination of the algorithm in Table 3.3, the solution of the given equation system has been computed.

Proof. The technique to solve a single equation is based on well-known approximation techniques [29, 88, 80], see Def. 3.2.11. Termination of this approximation means we have computed a stable solution to a single equation, which is also a least or greatest fixpoint resp. due to monotonicity of the first order boolean equations, see Lemma 3.2.5. Note that termination implies termination in a finite number of steps. The obtained solution can then be substituted in the lexicographically smaller equations of the equation system, as a result of Lemmas 3.2.9 and 3.2.10. Termination of the algorithm, therefore means we have correctly computed the solution to all equations in the equation system. □

Note that as it is undecidable whether a first order boolean equation system has a solution, the possible non-termination of our algorithm is unavoidable. Below, we provide three small examples, showing the application of the algorithm and its possible non-termination on systems that use data.

Example 3.3.2. Consider a counter that counts up to nine, starting from zero, and at nine cycles back to zero. Each time the counter increases, an *inc* event is issued. Upon reaching nine, the counter issues a *reset* event, signalling the counter has been reset to zero. A process algebraic description (in LPE form) of such a process is provided in Eqn. (3.13).

$$\begin{array}{rcl}\textbf{proc}\ C(n{:}\mathbb{N}) & = & [n \geq 9] ::\!\rightarrow \textit{reset} \cdot C(0) \\ & + & [n < 9] ::\!\rightarrow \textit{inc} \cdot C(n+1)\end{array} \tag{3.13}$$

Our goal is to verify whether it is possible to always execute a *reset* action. To this end, we specify the formula $\mu Y.[\mathsf{t}]Y \vee \langle \textit{reset} \rangle \mathsf{t}$. This basically expresses that on all infinite paths, eventually a *reset* action is executed. The first order boolean equation system for this expression is (after reduction) $\mu Z(n{:}\mathbb{N}) = (n \geq 9 \vee Z(n+1))$.

Following the algorithm, we first compute ψ_0 and ψ_1, being resp. f and $n \geq 9$. Then, we iterate until we end up with a formula $\psi_{10} = 0 \leq n$, which is equivalent to ψ_{11}. Since this is a stable solution of the equation, we can assess the truth of the equation system by substituting ψ_{10} for Z in our equation, thereby obtaining $\mu Z(n{:}\mathbb{N}) = \mathsf{t}$.

Example 3.3.3. As an example of a system with an infinite state-space, we consider a process that counts from zero to infinity, and reports its current state via an action *current*. A process algebraic description in LPE form is provided in Eqn. (3.14).

$$\textbf{proc}\ C(n{:}\mathbb{N}) = \textit{current}(n) \cdot C(n+1) \tag{3.14}$$

Given the simplicity of this process, it is unfortunate to find that with most current technologies, we cannot even automatically prove absence of deadlock for process C. Using our algorithm, this boils down to verifying $\nu X.\langle \mathsf{t} \rangle \mathsf{t} \wedge [\mathsf{t}]X$ on the process C. Following the translation, we derive the associated equation system $\nu Z(n{:}\mathbb{N}) = Z(n+1) \wedge \mathsf{t}$. Substituting t for $Z(n+1)$ immediately leads to the stable solution t.

Example 3.3.4. Consider a process *C* representing a counter that counts down from a randomly chosen natural number to zero and then randomly selects a new natural number, see Eqn. (3.15).

$$\begin{array}{rcl}\textbf{proc}\ C(n{:}\mathbb{N}) & = & \sum_{m:\mathbb{N}} [n = 0] ::\!\rightarrow \textit{reset} \cdot C(m) \\ & + & [n > 0] ::\!\rightarrow \textit{dec} \cdot C(n-1)\end{array} \tag{3.15}$$

Our goal is again to verify whether it is possible to always execute a *reset* action. This is again expressed as follows: $\mu Y.[\mathsf{t}]Y \vee \langle \textit{reset} \rangle \mathsf{t}$. The equation system for this expression is $\mu Z(n{:}\mathbb{N}) = n = 0 \vee Z(n-1)$.

The algorithm prescribes computing a stable solution for this equation. However, this computation does not terminate, as we end up with approximations ψ_k, where $\psi_k = n \leq k$. This means, we cannot find a ψ_j, such that $\psi_j = \psi_{j+1}$, and therefore, the algorithm does not terminate. However, it is straightforward to see that the minimal solution for this equation is $\mu Z(n{:}\mathbb{N}) = \mathsf{t}$.

3.4 Verification of Data-Dependent Systems in Practice

Based on our algorithm, described in the previous section, we have implemented a prototype of a tool. In this section, we briefly sketch this implementation, without going into too much detail. To test the applicability of the prototype, we have applied it on a large number of protocols. For brevity, we here report on the findings of only two smaller protocols with infinite state-spaces.

3.4.1 Implementation

The prototype implementation of our algorithm employs *Equational Binary Decision Diagrams* (EQ-BDDs) [56] for representing first order boolean expressions. These EQ-BDDs extend on standard BDDs [30] by explicitly allowing equality on nodes. We first define the grammar for EQ-BDDs.

Definition 3.4.1 (*Grammar for EQ-BDDs*).
We assume a set P of propositions and a set V of variables. The formulae we consider are given according to the grammar, given in Eqn. (3.16)

$$\Phi ::= \mathbf{0} \mid \mathbf{1} \mid \mathrm{ITE}(V = V, \Phi, \Phi) \mid \mathrm{ITE}(P, \Phi, \Phi) \tag{3.16}$$

The constants $\mathbf{0}$ and $\mathbf{1}$ represent *false* and *true*. An expression of the form $\mathrm{ITE}(\varphi, \psi, \xi)$ must be read as an *if-then-else* construct, i.e. $(\varphi \wedge \psi) \vee (\neg\varphi \wedge \xi)$, or, alternatively, $(\varphi \Rightarrow \psi) \wedge (\neg\varphi \Rightarrow \xi)$. For data variables d and e, and φ of the form $d = e$, the extension to EQ-BDDs is used, i.e. we explicitly use $\mathrm{ITE}(d = e, \psi, \xi)$ in such cases. Using the standard BDD and EQ-BDD encodings [30, 56], we can then represent all quantifier-free first order boolean expressions. The representation of expressions that contain quantifiers over finite domains is done in a straightforward manner, i.e. we construct explicit encodings for each distinct element in the domain. Expressions containing quantifiers over infinite domains are in general problematic when it comes to representation and calculations. The following theorem, however, identifies a number of cases in which we can deal with these.

Theorem 3.4.2 Quantification over data-types can be eliminated in the following cases: Suppose d does not occur in ψ. By abuse of notation, we write $d = e$, where we mean that the variable d takes on the value for variable e. We find:

- $\exists d{:}D.\mathrm{ITE}(d = e, \varphi, \psi) = \varphi[e/d] \vee \psi$ provided D contains at least two (distinct) elements.
- $\forall d{:}D.\mathrm{ITE}(d = e, \varphi, \psi) = \varphi[e/d] \wedge \psi$ provided D contains at least two (distinct) elements.
- $\exists d{:}D.\mathrm{ITE}(d = e_1, \varphi_1, \mathrm{ITE}(d = e_2, \varphi_2, \ldots, \mathrm{ITE}(d = e_n, \varphi_n, \psi) \ldots)) = \bigvee_{1 \le i \le n}((\bigwedge_{1 \le j < i} e_j \neq e_i) \wedge \varphi_i[e_i/d]) \vee \psi$ provided D contains at least one element not in $\{e_i | 1 \le i \le n\}$.
- $\forall d{:}D.\mathrm{ITE}(d = e_1, \varphi_1, \mathrm{ITE}(d = e_2, \varphi_2, \ldots, \mathrm{ITE}(d = e_n, \varphi_n, \psi) \ldots)) = \bigwedge_{1 \le i \le n}((\bigvee_{1 \le j < i} e_j = e_i) \vee \varphi_i[e_i/d]) \wedge \psi$ provided D contains at least one element not in $\{e_i | 1 \le i \le n\}$.

Proof. The identities follow directly from the observations that

- $\exists d{:}D.\mathrm{ITE}(d = e, \varphi, \psi) = \varphi[e/d] \vee \exists d{:}D(d \neq e \wedge \psi)$.

- $\forall d{:}D.\text{ITE}(d = e, \varphi, \psi) = \varphi[e/d] \wedge \forall d{:}D(d = e \vee \psi)$.

□

Note that the last two items of the theorem above actually say that if d only occurs in equations within a formula φ and the domain of D is sufficiently large, quantification over d can be removed, because each such formula can be brought into the form given above.

Even though Theorem 3.4.2 applies to a restricted class of first order boolean expressions, we find that in practice, it adds considerably to the verification power of the prototype implementation.

3.4.2 Example Verifications

We have used the prototype on several applications, including many communications protocols, such as the IEEE-1394 firewire, the sliding window protocol, the bounded retransmission protocol, etc. As an example of the capabilities, we here report on the use of our prototype on two small systems, viz. Lamport's Bakery Protocol [107], and the Alternating Bit Protocol [18]. Both systems have infinite state-spaces due to the use of infinite data domains, and the properties we are interested in are both liveness and safety properties. We first briefly introduce the two systems, and the properties we study.

Bakery Protocol The first example we consider is Lamport's Bakery protocol. A μCRL specification of this protocol is given in Table 3.4. The data-types are given as abstract data-types, but are omitted in this presentation. The bakery protocol we consider is restricted to two processes.

comm $\mathit{get}, \mathit{send} = c$

init $\partial_{\{\mathit{get},\, \mathit{send}\}}(P(\mathsf{t}) \| P(\mathsf{f}))$

proc $P(b{:}\mathbb{B}) = \mathit{request}(b) \cdot P_0(b, 0) + \mathit{send}(b, 0) \cdot P(b)$

proc $P_0(b{:}\mathbb{B}, n{:}\mathbb{N}) = \sum_{m:\mathbb{N}} \mathit{get}(\neg b, m) \cdot P_1(b, m+1) + \mathit{send}(b, n) \cdot P_0(b, n)$

proc $P_1(b{:}\mathbb{B}, n{:}\mathbb{N}) =$
$\sum_{m:\mathbb{N}} \mathit{get}(\neg b, m) \cdot (C_1(b, n) \lhd n < m \vee m = 0 \rhd P_1(b, n)) + \mathit{send}(b, n) \cdot P_1(b, n)$

proc $C_1(b{:}\mathbb{B}, n{:}\mathbb{N}) = \mathit{enter}(b) \cdot C_2(b, n) + \mathit{send}(b, n) \cdot C_1(b, n)$

proc $C_2(b{:}\mathbb{B}, n{:}\mathbb{N}) = \mathit{leave}(b) \cdot P(b) + \mathit{send}(b, n) \cdot C_2(b, n)$

Table 3.4: Lamport's Bakery Protocol

An informal explanation of the protocol is as follows. A process, waiting to enter its critical section can choose a number, larger than any other number already chosen. Then, the process with the lower number is allowed to enter the critical section before the process with the larger number.

Given the unbounded growth of the numbers that can be chosen, the protocol clearly has an infinite state-space. Hence, verification of the bakery protocol is usually performed on an altered version, abstracting in some way from these numbers. Our techniques, however, are immediately applicable. Below, we list a number of key properties we verify for the bakery protocol.

1. No deadlock can occur, i.e. in every reachable state of the protocol, an action is enabled,

2. All processes requesting a number can eventually enter the critical section,

3. All processes requesting a number inevitably enter the critical section.

The results for the verification of these properties are listed in Table 3.5. The second and third property deserve some extra attention, as their difference is quite subtle; it is best compared to the difference between “may” and “must”. The second property states that if a process requests a number, there is a path leading towards a state in which a process may gain access to the critical section. The third property states that if a process requests a number, all paths inevitably lead to states in which the process must gain access to the critical section. Note that requesting a number (using action *request*) is not a sufficient condition for entering the critical section, as this is only guaranteed when the process has also received its number (using action *get*). This explains why the third property does not hold: it can be the case that the request of the number is not followed by the receiving of a number, in which case the other process can infinitely often access the critical section.

Nr.	Formal Property	Satisfied	Time
1.	$\nu X.([\mathsf{t}]X \wedge \langle \mathsf{t} \rangle \mathsf{t})$	yes	2sec
2.	$\nu X.([\mathsf{t}]X \wedge \forall b{:}\mathbb{B}.[\mathit{request}(b)]\mu Y.\langle \mathsf{t} \rangle Y \vee \langle \mathit{enter}(b) \rangle \mathsf{t})$	yes	60sec
3.	$\nu X.([\mathsf{t}]X \wedge \forall b{:}\mathbb{B}.[\mathit{request}(b)]\mu Y.(([\mathsf{t}]Y \wedge \langle \mathsf{t} \rangle \mathsf{t}) \vee \langle \mathit{enter}(b) \rangle \mathsf{t}))$	no	5sec

Table 3.5: Verification results of the Bakery protocol. All computations were performed on a 1 GHz Intel Pentium III processor with 512Mb main memory running Linux version 2.4. The field “Time” states the amount of computation time needed to perform the verification.

Alternating Bit Protocol The *Alternating Bit Protocol* (ABP, see e.g. [18]) is a basic communications protocol utilising two channels. A sender sends a message, tagged with a bit, via an unreliable channel. It repeatedly resends this message (including the bit), until it receives an acknowledgement (with the right bit) from the receiver, via the other channel. It then starts the entire procedure again with a new message, and inverts the bit it sends along with the message.

The ABP is a famous communications protocol, and is often used to illustrate that a formalism or technique is capable of dealing with real systems of small to medium size. When applying well-established, fully-automatic techniques, the data that is transmitted in this (and other) communications protocols, has to be fixed. Here, we show that, with the use of our prototype, no alterations to the ABP are necessary, and the messages we transmit are indeed arbitrarily chosen from an infinite set of messages. Communications protocols usually have an external behaviour,

comm $r2, s2 = c2$
$r3, s3 = c3$
$r5, s5 = c5$
$r6, s6 = c6$

init $\partial_{\{r2,r3,r5,r6,s2,s3,s5,s6\}}(S \| K \| L \| R)$

proc $S = S(0) \cdot S(1) \cdot S$

proc $S(n{:}\textit{\textbf{bit}}) = \sum_{d:D} r1(d) \cdot S(d, n)$

proc $S(d{:}D, n{:}\textit{\textbf{bit}}) = s2(d, n) \cdot ((r6(\textit{\textbf{invert}}(n)) + r6(e)) \cdot S(d, n) + r6(n))$

proc $R = R(1) \cdot R(0) \cdot R$

proc $R(n{:}\textit{\textbf{bit}}) =$
$(r3(e) + \sum_{d:D} r3(d, n)) \cdot s5(n) \cdot R(n) + \sum_{d:D} r3(d, \textit{\textbf{invert}}(n)) \cdot s4(d) \cdot s5(\textit{\textbf{invert}}(n))$

proc $K = \sum_{d:D} \sum_{n:\textit{\textbf{bit}}} r2(d, n) \cdot (i \cdot s3(d, n) + i \cdot s3(e)) \cdot K$

proc $L = \sum_{n:\textit{\textbf{bit}}} r5(n) \cdot (i \cdot s6(n) + i \cdot s6(e)) \cdot L$

Table 3.6: Alternating Bit Protocol

similar to the behaviour of a buffer, i.e. messages sent at one end are eventually received at the other end. The ABP is no exception to this rule. The properties we verified for ABP are listed below.

1. No deadlock can occur,
2. A message that is sent always eventually is received,
3. The protocol does not create messages,
4. The protocol does not duplicate messages.

The two latter properties state that the protocol does not allow for any miracles to happen. The results of the verification of these properties are listed in Table 3.7.

3.5 Closing Remarks

Related Work In this thesis, we focused on the automatic verification of large and infinite state, data-dependent systems. In the fully automatic verification of systems, we can distinguish

Nr.	Formal Property	Satisfied	Time
1.	$\nu X.([\mathsf{t}]X \wedge \langle\mathsf{t}\rangle\mathsf{t})$	yes	2sec
2.	$\nu X.([\mathsf{t}]X \wedge \forall d{:}D.[r1(d)]\mu Y.\langle\mathsf{t}\rangle Y \vee \langle s4(d)\rangle\mathsf{t})$	yes	15sec
3.	$\nu X.(\forall d{:}D.([\neg r1(d)]X \wedge [s4(d)]\mathsf{f}))$	yes	60sec
4.	$\nu X.([\mathsf{t}]X \wedge \forall d{:}D.[r1(d)]\nu Y.([\neg r1(d) \vee s4(d)]Y \wedge [r1(d)]\mathsf{f}))$	yes	5sec

Table 3.7: Verification results of the Alternating Bit Protocol. All computations were performed on a 1 GHz Intel Pentium III processor with 512Mb main memory running Linux version 2.4. The field "Time" states the amount of time needed to perform the verification.

between three different approaches.

The first approach is to develop dedicated techniques to deal with specialised classes of systems. Such classes contain communication protocols (e.g. regular expressions [1], queue representations [23]) process networks (e.g. Presburger arithmetic [32]) and parameterised systems (e.g. counter abstraction [103]). This clearly contrasts to our work, as we do not restrict the class of systems we consider upfront.

The second approach is to deal with a restricted class of properties that can be verified. A promising techniques in this direction is the *Counter arithmetic with Lambda expressions and Uninterpreted functions* (CLU) by Bryant *et al* [31]. CLU is very general in that it can be used to model both data and control, and in [31], it is shown to be decidable. The tool, based on CLU – UCLID – is restricted to dealing with safety properties only. Our approach is more general, as it allows safety, liveness and fairness properties to be verified automatically. Moreover, CLU is restricted to the quantifier-free fragment of first order logic, whereas the logic we use in our approach employs the full first order logic.

The third approach does in general not lay any restrictions on the class of systems, or on the class of properties that can be examined. This is in line with the approach we follow. Mateescu [93, 92] defines the language XTL, which is the name for a specification language and a model-checker. XTL is a quite involved functional programming language that allows for the specification of temporal logic properties over states and transitions and involves data values. Other temporal logics, such as CTL, ACTL and the μ-calculus have been implemented in XTL. Since the tool works on an explicit representation of the state-space, there is a restriction that the data types cannot be infinite. As such, the approach we followed in this chapter is more liberal than this approach.

The tool EVALUATOR 3.0, developed by Mateescu and Sighireanu [94, 91] is an on-the-fly model checker for the regular alternation-free μ-calculus [92]. The machinery of EVALUATOR 3.0 is based on *boolean equation systems*, and is comparable to our approach. Although the regular alternation-free μ-calculus allows temporal logic properties involving data, the current version of the tool does not support the data-based version of this language. It is well imaginable that this tool can be extended with our techniques.

The technique by Bultan *et al.* [32] seems to be able to produce results that are comparable to ours. Their techniques, however, are entirely different from ours. In fact, their approach is similar to the approach used by Alur *et al.* [6] for hybrid systems. It uses affine constraints on integer variables, logical connectives and quantifiers to symbolically encode transition relations and sets of states. The logic, used to specify the properties is a CTL-style logic. In order to guarantee termination, they introduce conservative approximation techniques that may yield "false negatives", which always converges.

Discussion We developed a technique for model checking systems that depend on (possibly infinite) data-types. The techniques and algorithm we used, are based upon the techniques and algorithm, described by e.g. Mader [88]. We utilise equation systems as an intermediate formalism to which we translate both our system description (given in μCRL) and a property description (given in a first order modal μ-calculus). Given that the problem in general is not decidable, we have assessed the applicability of our solution on a large number of cases, both finite and infinite state systems. In Section 3.4, we have reported on the results obtained for two small systems with infinite state-spaces, showing that the tool indeed functions as expected.

The experiences we obtained by verifying properties in the other systems (e.g. the Bounded Retransmission Protocol) show that the tool indeed sometimes fails to terminate, but in general, termination is achieved in an acceptable run-time. Indeed, in several instances, we have been able to use our prototype in situations where existing, well-established, tool-sets failed to produce the result. Slightly surprising is one such instance, viz. a subsystem of the *EUV Wafer Stepper Machine* [90, 2] of ASML, for which we had two different specifications. Using our prototype, we were able to verify properties of the specification given in [90] where the general μCRL tool-suite [21] (using the Cæsar-Aldébaran [45, 46] front-end) failed to even build an internal representation of the state-space. The specification, given in [2] was no problem for the general μCRL tool-suite, yet proved troublesome for our prototype.

We conclude that the prototype tool we have developed is a welcome addition to the μCRL tool-suite. However, we feel it is not suited for inclusion yet. At present, the prototype implementation builds an internal representation of an equation system and immediately tries to calculate its solution. We envision the development of a compiler that combines a μCRL process and a first order modal μ-calculus formula into an equation system. It is our intention to then introduce new tools that operate on first order boolean equation systems, including a tool that, when fed an equation system, computes its solution.

Summary Summarising, we find that the verifications take in many cases an acceptable run-time, even though for systems with finite state spaces, our prototype is often outperformed by most well-established tool-suites. However, we expect some improvements can still be made on the prototype. More importantly, as we have demonstrated in Section 3.4, we are able to automatically verify properties of systems with infinite state-spaces in a reasonable time. Also, we have successfully applied our prototype on a system with an extremely large state-space, for which established techniques failed to calculate the exact state-space. Since this is where the current state-of-the-art technology breaks down, our technique is clearly an improvement on the current technology.

The prototype implementation, however, also revealed a number of new issues to be resolved. We were not able to prove absence of deadlock of the Bounded Retransmission Protocol [55], with arbitrary bound on the number of retransmissions. As it turns out, the current rewrite strategy, used for rewriting the abstract data types is not particularly well-suited for dealing with this case. The possible solutions to this problem may lie in considering e.g. *Associative-Commutative* rewriting (see e.g. [39]).

Still, several other issues remain to be investigated. First, in [51], Groote and Mateescu provide four deduction rules for manually establishing the truth or falsity of a formula on an infinite state-space. It is interesting to see if some, or parts of these proof rules can be automated, thereby solving problems that our algorithm cannot deal with. Second, techniques, such as developed by Pnueli *et al* [103], and Bryant *et al* [31, 115] may be incorporated to increase the success rate of the algorithm we proposed. Third, the prototype has only limited diagnostic features. It requires additional research to obtain more meaningful diagnostics, such as failure traces, to increase the usability of the prototype.

Chapter 4

Process-Behaviour Restriction

The possibility of verifying (models of) systems has silently become one of the most prominent reasons for applying formal methods in computer science. Techniques for verification are applied *a posteriori*, giving additional insight in a system, on top of the confidence obtained by proving required properties of the system holds. Opposed to verification, synthesis is a technique that is applied *a priori*. Unfortunately, existing techniques for synthesising are often of far greater complexity than verification techniques, leaving their applicability somewhat debatable. In this chapter, we discuss a synthesis technique that is arguably still useful in practice and relieves designers of complex systems of some of the burdens of carefully bookkeeping and administration.

The synthesis technique discussed in this chapter, hereafter referred to as *process-behaviour restriction*, deals with the restriction of the *uninitialised* system's behaviour such that it satisfies properties that can be specified as predicates on the system's state space. By uninitialised, we mean that the initial state of the system is not pre-determined but can potentially be any value in some domain satisfying all required properties. Thus, an uninitialised system is a parameterised system where the initial state is not yet fixed. This has a definite advantage, since it allows us to deal with systems that are used in a wide range of environments, each requiring the system to start in different initial states. Noteworthy examples of such systems are plug and play *components*, or processes that have to be fault tolerant, and should operate reliable under any given circumstance. Note that many popular modelling languages, such as timed automata [8] and hybrid automata [62] also allow systems with multiple initial states.

Rather than studying the multi-step control problem (studied in e.g. [60, 105, 117]), we here address the single-step control problem (studied in [3, 4]) in this chapter. The multi-step problem is often solved by iteratively solving the single-step control problem. It turns out that the single-step control problem in the presence of multiple initial states is already quite challenging.

Using invariants [20] for verification purposes is a well-established technique with different types of application (see e.g. Chapter 7 and the use of invariants in the Cones and Foci technique [54]). In this chapter, invariants arise as a result of the restrictions applied to a process. The properties that are expressed by these invariants are restricted to safety properties; liveness properties are sometimes obtained as a side-effect, but in general cannot be expressed by an invariant. However, studying a process that has been restricted to satisfy an invariant, using e.g. the techniques developed in the previous chapter, may very well reveal a clash between the (satisfied) safety properties and the (desired) liveness property. Hence, it pays to study the states of a process that are considered safe in isolation.

Restriction of a system's behaviour is possible by specifying the conditions under which the execution of an action is allowed. Not all actions can be restricted in this sense, so a natural distinction we can make is to distinguish *controllable* from *uncontrollable* actions. This terminology is used also in e.g. [105, 106]. We first study the process-behaviour restriction problem for a class of processes for which actions are not parameterised with data. We then turn to the class of processes that can also communicate data with its environment, thereby increasing the complexity. In Section 4.3, we investigate under which conditions we can guarantee absence of deadlock for the restricted process.

4.1 Restriction for Basic Processes

Basic processes are untimed processes that have unparameterised actions. The number of systems that can adequately be described is rather restricted, however, non-determinism in itself already is sufficient for non-trivial solutions to the process-behaviour restriction problem.

The basic processes we consider can all be represented by means of *Basic Linear Process Equations* (BLPE). The form of a BLPE is in essence a restriction of an LPE, already discussed in Chapter 2. We focus on processes that can run indefinitely, or terminate unsuccessfully by deadlocking.

Definition 4.1.1 (*Basic Linear Process Equation*).
A process equation is called a *Basic Linear Process Equation* if it is of the form of Eqn. (4.1).

$$P(d{:}D) = \sum_{i\in\mathcal{I}} \sum_{e_i:D_i} [b_i(d,e_i)] :: \rightarrow a_i \cdot P(g_i(d,e_i))) \tag{4.1}$$

Here, the function g_i encodes the *next-state* relation. The enabledness of action a_i is governed by a predicate b_i.

The parameter-space (or *state-space*) of a (Basic) Linear Process Equation $P{:}D \to \mathbb{P}$, is given by the set D. Dependent on the initial values for the process P, either a proper subset of D, or the entire set D is the reachable parameter-space of the process P. Invariants express properties over the reachable parameter-space of a BPLE, or, seen from a different perspective, invariants *characterise* the reachable parameter-space. Formally, an invariant of a (B)LPE $P{:}D \to \mathbb{P}$ is a predicate $\iota{:}D \to \mathbb{B}$, such that for all i and e_i, $\iota(d) \wedge b_i(d,e_i)$ it follows that $\iota(g_i(d,e_i))$ holds. The *property set* $\mathcal{I}_\iota$ is then the subset of D, satisfying ι, i.e. for all $d \in D$, we have $d \in \mathcal{I}_\iota$ iff $\iota(d)$ holds. For a given parameter $d \in D$, we write $P(d) \models \iota$ iff $\iota(d)$ holds and ι is an invariant of process P. If there can be no confusion, we omit the property ι when referring to the property set, and write $\mathcal{I}$ rather than $\mathcal{I}_\iota$.

Remark 4.1.2. *The definition of an invariant was taken from [20], where invariants are used in the context of process algebras with data, and in fact, introduced in μCRL. It turns out that these types of invariants are also known as inductive invariants in the literature.*

In this chapter, we investigate the restricting of a process' behaviour by defining which states of a process may be reached. For this, we use invariants. By means of synchronising on a set of actions, we can allow or disallow possible next states. The actions on which synchronisation is allowed are referred to as *controllable actions*. Similarly, the actions that we cannot prevent from happening, or observe as having occurred, are *uncontrollable actions*. The constant δ, representing inaction, is neither controllable, nor uncontrollable. Note that for controllable actions, we are allowed to define the communications function γ (see Chapter 2), whereas the communications function is undefined for uncontrollable actions. From here-on, actions (different from δ) are either controllable or uncontrollable.

As we mentioned, the process-behaviour restriction, considered in this chapter, is achieved by controlling the enabledness of controllable actions. Uncontrollable actions are transparent to process-behaviour restriction, and thus should be taken into account when controlling the enabledness of controllable actions. We define two relations, viz. *autonomous reachability* and *controllable autonomous reachability*.

Definition 4.1.3 (*Autonomous Reachability, Controllable Autonomous Reachability*).
The *autonomous reachability* relation $\Omega_A{:}2^D \to 2^D$ relates a set of states to the set of states that are reachable from these states via the execution of zero or more actions from the set A (not containing δ). The relation $\Omega_A = \bigcup_{i=0}^{\omega} \Omega_A^i$, defined inductively by Eqn. (4.2), can be used to capture the uncontrollability of actions by merging states that can be reached via enabled uncontrollable actions.

$$\begin{array}{ll} \Omega_A^0(\hat{d}) & = \emptyset \\ \Omega_A^{n+1}(\hat{d}) & = \hat{d} \cup \Omega_A^n(\{g_i(d,e_i) \in D \mid i \in I \wedge a_i \in A \wedge e_i{:}D_i \wedge d \in \hat{d} \wedge b_i(d,e_i)\}) \end{array} \tag{4.2}$$

Controllable Autonomous Reachability $\Omega_{c,A}{:}2^D \to 2^D$ relates a set of states to the set of states that are reachable autonomously from these states by first executing an action c and then zero or more actions from the set A. The action c typically is a controllable action, whereas the set A usually is taken to be the set of uncontrollable actions, not containing δ.

$$\Omega_{c,A}(\hat{d}) = \Omega_A(\{g_i(d,e_i) \in D \mid i \in I \wedge a_i = c \wedge d \in \hat{d} \wedge e_i{:}D_i \wedge b_i(d,e_i)\}) \tag{4.3}$$

Property 4.1.4 For arbitrary sets $A, \hat{d}$ and $\hat{e}$, and (controllable) actions c, we have

1. $\Omega_A(\hat{d} \cup \hat{e}) = \Omega_A(\hat{d}) \cup \Omega_A(\hat{e})$,
2. $\hat{d} \subseteq \hat{e}$ then also $\Omega_A(\hat{d}) \subseteq \Omega_A(\hat{e})$,
3. $\hat{d} \subseteq \hat{e}$ then also $\Omega_{c,A}(\hat{d}) \subseteq \Omega_{c,A}(\hat{e})$.

Proof. The proof of Property 4.1.4 proceeds by induction on the approximation of Ω_A. We first prove that for all $n \in \mathbb{N}$, $\Omega_A^n(\hat{d} \cup \hat{e}) = \Omega_A^n(\hat{d}) \cup \Omega_A^n(\hat{e})$.

1. We have $\Omega_A^0(\hat{d} \cup \hat{e}) = \emptyset = \Omega_A^0(\hat{d}) \cup \Omega_A^0(\hat{e})$,
2. Suppose $\Omega_A^n(\hat{d} \cup \hat{e}) = \Omega_A^n(\hat{d}) \cup \Omega_A^n(\hat{e})$. By definition, we have $\Omega_A^{n+1}(\hat{d} \cup \hat{e})$ is equivalent to $(\hat{d} \cup \hat{e}) \cup \Omega_A^n(\{g_i(d,e_i) \in D \mid i \in I \wedge a_i \in A \wedge e_i{:}D_i \wedge d \in (\hat{d} \cup \hat{e}) \wedge b_i(d,e_i)\})$. Using induction, this is easily seen to be equivalent to $\Omega_A^{n+1}(\hat{d}) \cup \Omega_A^{n+1}(\hat{e})$.

Now, suppose we have $\hat{d} \subseteq \hat{e}$, then there exists a $\hat{f}$, such that $\hat{e} = \hat{d} \cup \hat{f}$. We then know that $\Omega_A(\hat{e}) = \Omega_A(\hat{d} \cup \hat{f}) = \Omega_A(\hat{d}) \cup \Omega_A(\hat{f}) \supseteq \Omega_A(\hat{d})$, which proves the second property. The third property follows immediately from the second property. □

The actual process-behaviour restriction of a process is obtained by synchronising this process with a ι-*controller*, where ι represents the property that must be ensured. The actual definition of a ι-controller is based on the process and the predicate ι. We sometimes omit the ι if it the property is immaterial or arbitrarily chosen and use the word *controller* to denote a possible ι-controller.

Definition 4.1.5 (ι-*Controller*).
Let $P{:}D \to \mathbb{P}$ be a process with controllable action set A_c and uncontrollable action set A_u; let $\iota{:}D \to \mathbb{B}$ a predicate. A process $C{:}D' \to \mathbb{P}$, with action set $\overline{A}_c$ is a ι-controller iff there exists a function $f{:}D \to D'$, an *initial safe-set* $S \subseteq D$ and $\iota'{:}D \times D' \to \mathbb{B}$, where $\iota'(d, d') = \iota(d)$ for all d' (i.e. ι is the projection of ι'), such that for all $d \in S$, relation (4.4) holds.

$$\partial_{A_c \cup \overline{A}_c}(P(d) \| C(f(d))) \models \iota' \tag{4.4}$$

Notation 4.1.6. We use the set $\overline{A}_c$ to denote the set of co-controllable actions, used to signal whether the execution of a controllable action is allowed, i.e. $\overline{A}_c = \{\overline{a} \mid a \in A_c\}$. Communication between a controllable action and a co-controllable action is defined by $\gamma(a, \overline{a}) = \mathsf{a}$ for all $a \in A_c, \overline{a} \in \overline{A}_c$.

In the remainder of this chapter, we use a specific ι-controller, which is referred to as the ι-*restrictor* for a given process and property ι. Before arriving at the definition of the ι-restrictor, we define the notion of ι-*admissible states*, capturing the set of states that are guaranteed to satisfy a predicate, regardless of the uncontrollable activity that is possible from these states.

Definition 4.1.7 (ι-*Admissible States*).
Let $P{:}D \to \mathbb{P}$ be a process with uncontrollable action set A_u, and let $\iota{:}D \to \mathbb{B}$ be a predicate (with associated property set $\mathcal{I}$). A state $d \in D$ is said to be admissible iff $\Omega_{A_u}(\{d\}) \subseteq \mathcal{I}$.

Definition 4.1.8 (ι-*Restrictor*).
Let $P{:}D \to \mathbb{P}$ be a Basic Linear Process Equation with controllable action set A_c and uncontrollable action set A_u, denoting a system's behaviour. Let $\iota{:}D \to \mathbb{B}$ be a predicate on the parameter-space of P, representing a desired property of the system. Define the initial safe-set $S \subseteq D$ as the set $\{d \in \mathcal{I} | \Omega_{A_u}(\{d\}) \subseteq \mathcal{I}\}$. The ι-*restrictor* $C{:}2^D \to \mathbb{P}$ for this system is defined by Eqn. (4.5).

$$C(\hat{d}{:}2^D) = \sum_{a \in A_c} [\Omega_{a,A_u}(\hat{d}) \subseteq \mathcal{I}] ::\to \overline{a} \cdot C(\Omega_{a,A_u}(\hat{d})) \tag{4.5}$$

Operationally, the ι-restrictor checks if the execution of a controllable action, say action a, is an allowed behaviour when a process is in one of the states in $\hat{d}$. If so, the action $\overline{a}$ is enabled. If the action a is not at all possible (i.e. $\Omega_{a,A_u}(\hat{d}) = \emptyset$), the ι-restrictor allows all occurrences of action a (being none). Alternatively, we could in this case "disallow" the action a by requiring the condition $\emptyset \subset \Omega_{a,A_u}(\hat{d}) \subseteq \mathcal{I}$ rather than $\Omega_{a,A_u}(\hat{d}) \subseteq \mathcal{I}$. Using this latter condition, or the one used in Eqn. (4.5) has no consequences for the envisioned use of ι-restrictors.

Provided that the ι-restrictor for a process P is instantiated with the right parameters, the synchronous product of the ι-restrictor and the process P can be shown to enforce process-behaviour restriction. This is formalised as follows.

Lemma 4.1.9.
Let $P{:}D \to \mathbb{P}$ be a process, $\iota{:}D \to \mathbb{B}$ be a predicate, and $C{:}2^D \to \mathbb{P}$ be the ι-restrictor as defined in Def. 4.1.8. Define $PC(d,\hat{d}) = \partial_{A_c \cup \overline{A}_c}(P(d) \| C(\hat{d}))$. Define the communication function γ as $\gamma(a, \overline{a}) = \mathsf{a}$ for all $a \in A_c$. Then, process $PC{:}D \times 2^D \to \mathbb{P}$ satisfies the following invariants, provided they are satisfied in the initial states:

1. $\Omega_{A_u}(\{d\}) \subseteq \hat{d} \wedge d \in \hat{d}$,

2. $\hat{d} \subseteq \mathcal{I}$.

Proof. Let process $PC{:}D \times 2^D \to \mathbb{P}$ be as defined by Lemma 4.1.9.

1. Assume $\Omega_{A_u}(\{d\}) \subseteq \hat{d} \wedge d \in \hat{d}$, and assume process PC executes an action a'. We distinguish the two types of actions, viz. controllable and uncontrollable actions.

   (a) Suppose action a is due to the communication of a controllable action a' and a co-controllable action $\overline{a}'$. Then, we need to prove that $\Omega_{A_u}(\{g_i(d,e_i)\}) \subseteq \Omega_{a',A_u}(\hat{d})$ and $g_i(d,e_i) \in \Omega_{a',A_u}(\hat{d})$ hold for all $e_i{:}D_i$, such that $b_i(d,e_i)$ for all $i \in I$ for which $a_i = a'$, given $\Omega_{a',A_u}(\hat{d}) \subseteq \mathcal{I}$. The first part is trivial, since $\Omega_{a',A_u}(\hat{d}) = \Omega_{A_u}(\{g_i(d,e_i') | i \in I \wedge a_i = a' \wedge d \in \hat{d} \wedge e_i'{:}D_i \wedge b_i(d,e_i')\}) \supseteq \Omega_{A_u}(\{g_i(d,e_i)\})$ for $d \in \hat{d}$ and $i \in I$ such that $a_i = a'$ (see Property 4.1.4). Similarly, the second part holds because $g_i(d,e_i) \in \Omega_{A_u}(\{g_i(d,e_i') | i \in I \wedge a_i = a' \wedge d \in \hat{d} \wedge e_i'{:}D_i \wedge b_i(d,e_i')\})$ and $d \in \hat{d}$.

   (b) Suppose action a is due to an uncontrollable action a of process P. Then, we must prove that $\Omega_{A_u}(\{g_i(d,e_i)\}) \subseteq \hat{d}$ and $g_i(d,e_i) \in \hat{d}$ for all $i \in I$ such that $a_i = a$. Since we know that for all $i \in I$ for which $a_i = a$, $\Omega_{A_u}(\{g_i(d,e_i)\}) \subseteq \Omega_{A_u}(\{d\})$, we also know the desired $\Omega_{A_u}(\{g_i(d,e_i)\}) \subseteq \hat{d}$ for all $i \in I$ such that $a_i = a$. Moreover, $g_i(d,e_i) \in \hat{d}$ follows from $g_i(d,e_i) \in \Omega_{A_u}(\{g_i(d,e_i)\}) \subseteq \hat{d}$.

2. Assume $\hat{d} \subseteq \mathcal{I}$, and assume process $PC{:}D \times 2^D \to \mathbb{P}$ executes an action.

   (a) Suppose this action is due to a communication of a controllable action a' and a co-controllable action $\overline{a}'$. Then, apparently $\Omega_{a',A_u}(\hat{d}) \subseteq \mathcal{I}$ holds, otherwise the action could not have been executed.

   (b) Suppose action a is an uncontrollable action. Then, this action has no effect on the value for $\hat{d}$, and thus, we still know $\hat{d} \subseteq \mathcal{I}$ afterwards.

□

Lemma 4.1.9 provides the basis for the proof that the ι-restrictor as defined in Def. 4.1.8 is indeed a ι-controller.

Theorem 4.1.10 (*A ι-Restrictor is a ι-Controller*).
Let $P{:}D \to \mathbb{P}$ be a process, $\iota{:}D \to \mathbb{B}$ be a predicate, and $C{:}2^D \to \mathbb{P}$ be the ι-restrictor as defined in Def. 4.1.8. Then, for all $d \in \mathcal{I}$, for which $\Omega_{A_u}(\{d\}) \subseteq \mathcal{I}$, identity (4.6) holds.

$$\partial_{A_c \cup \overline{A}_c}(P(d) \| C(\Omega_{A_u}(\{d\}))) \models \iota' \tag{4.6}$$

Proof. Let $d \in \mathcal{I}$, such that $\Omega_{A_u}(\{d\}) \subseteq \mathcal{I}$ holds. By Lemma 4.1.9, we know that $d \in \hat{d}$ and $\hat{d} \subseteq \mathcal{I}$, and therefore, we also know that $d \in \mathcal{I}$ is an invariant for the process $PC{:}D \times 2^D \to \mathbb{P}$, thus ι' (and therefore ι) is satisfied as a property of process PC. □

The ι-restrictor as defined by Def. 4.1.8 is not the only ι-controller, as we can prove that, e.g. the process δ is a ι-controller for arbitrary properties and processes. However, the ι-controller defined by Def. 4.1.8 is the least restrictive of all ι-controllers, in the sense that it allows behaviours other ι-controllers do not allow. We formalise the notion of restrictiveness and in Theorem 4.1.12 we show that the ι-restrictor of Def. 4.1.8 satisfies this definition.

Definition 4.1.11 (*Restrictiveness*).
Let $P{:}D \to \mathbb{P}$ be given by a Basic Linear Process Equation and let $\iota{:}D \to \mathbb{B}$ be a predicate. Let $C'{:}D' \to \mathbb{P}$ and $C''{:}D'' \to \mathbb{P}$ be arbitrary ι-controllers for process P, with initial safe-sets S' and S'' and functions $f'{:}D \to D'$ and $f''{:}D \to D''$. Then, ι-controller C' is *less restrictive* than C'' iff there exists a simulation relation $\mathcal{R}$ (recall Def. 2.2.9), such that identity (4.7) holds.

$$\partial_{A_c \cup \overline{A}_c}(P(d) \| C'(f'(d)))\ \mathcal{R}\ \partial_{A_c \cup \overline{A}_c}(P(d) \| C''(f''(d))) \tag{4.7}$$

A ι-controller C is called a *least restrictive ι-controller* iff for all ι-controllers C', C is less restrictive than C'.

Theorem 4.1.12 (*The ι-Restrictor is a Least Restrictive ι-Controller*).
Let $P{:}D \to \mathbb{P}$ be a Basic Linear Process Equation and let $\iota{:}D \to \mathbb{B}$ be a predicate. Let $C{:}2^D \to \mathbb{P}$ be the ι-restrictor as defined in Def. 4.1.8. Then, C is a least restrictive ι-controller.

Proof. Eqn. (4.8) provides the simulation relation $\mathcal{R}$ for an initial state $d_0 \in S$; we subsequently prove $\mathcal{R}$ is indeed a simulation relation.

$$\begin{aligned}\mathcal{R} = \ & \{(\partial_{A_c \cup \overline{A}_c}(P(d) \| C'(d')), \partial_{A_c \cup \overline{A}_c}(P(d) \| C(\hat{d})))\\ & \mid \exists_{\sigma \in (\mathbf{A}_c \cup A_u)^*}\ \partial_{A_c \cup \overline{A}_c}(P(d_0) \| C'(f(d_0))) \overset{\sigma}{\twoheadrightarrow} \partial_{A_c \cup \overline{A}_c}(P(d) \| C'(d')) \wedge \\ & \qquad \partial_{A_c \cup \overline{A}_c}(P(d_0) \| C(\Omega_{A_u}(\{d_0\}))) \overset{\sigma}{\twoheadrightarrow} \partial_{A_c \cup \overline{A}_c}(P(d) \| C(\hat{d}))\}\end{aligned} \tag{4.8}$$

For $\mathcal{R}$ to be a simulation relation, we must relate initial states and provide single-step simulation.

1. Let $d_0 \in S$, then also $\partial_{A_c \cup \overline{A}_c}(P(d_0) \| C'(f(d_0))) \mathcal{R} \partial_{A_c \cup \overline{A}_c}(P(d_0) \| C(\Omega_{A_u}(\{d_0\})))$. Suppose this is not true, then this means $\Omega_{A_u}(\{d_0\}) \not\subseteq \mathcal{I}$, meaning that $P(d_0)$ can execute a sequence of uncontrollable actions leading to a state $P(d_1)$ violating ι. If this is the case, then controller C' has no means of preventing this sequence and hence C' is not a controller, which is a contradiction.

2. Suppose $\partial_{A_c \cup \overline{A}_c}(P(d) \| C'(d')) \mathcal{R} \partial_{A_c \cup \overline{A}_c}(P(d) \| C(\hat{d}))$. Furthermore, assume we also have $\partial_{A_c \cup \overline{A}_c}(P(d) \| C'(d')) \xrightarrow{a} \partial_{A_c \cup \overline{A}_c}(P(e) \| C'(e'))$. Then, for some $\hat{e} \in 2^D$, we also have $\partial_{A_c \cup \overline{A}_c}(P(d) \| C(\hat{d})) \xrightarrow{a} \partial_{A_c \cup \overline{A}_c}(P(e) \| C(\hat{e}))$. Suppose this is not the case. We now can distinguish two cases: either action a is controllable or action a is uncontrollable. We deal with these cases separately.

   - Suppose action a is uncontrollable. Then, by construction of the ι-restrictor, we have $e \in \hat{d}$, since $\Omega_{A_u}(\{d\}) \subseteq \hat{d}$ (see Lemma 4.1.9). Therefore, action a cannot be uncontrollable.

- Suppose action a is controllable. Then, the only situation for the ι-restrictor to disallow the action a is whenever $\Omega_{a,A_u}(\hat{d}) \nsubseteq \mathcal{I}$. This, means there is a trace σa that leads to a situation in which the invariant ι is violated. But if that is the case, then controller C' also has no means of preventing this trace, and therefore is not a controller. Therefore, action a cannot be controllable.

Both cases lead to a contradiction; thence $\partial_{A_c \cup \overline{A}_c}(P(d) \| C(\hat{d})) \xrightarrow{a} \partial_{A_c \cup \overline{A}_c}(P(e) \| C(\hat{e}))$, and, moreover, $\partial_{A_c \cup \overline{A}_c}(P(e) \| C'(e')) \mathcal{R} \partial_{A_c \cup \overline{A}_c}(P(e) \| C(\hat{e}))$.

□

4.2 Restriction for Full Processes

Absence of parameterised actions makes basic processes, as dealt with in Section 4.1, unsuitable for many applications. Often, parameterised actions are used in a context where data plays a decisive role in the behaviour of a process, e.g. communications protocols.

In this section, we extend the theory of process-behaviour restriction to deal with processes with untimed but parameterised actions. Parameterised actions are a source of additional non-determinism in a system's behaviour. Managing this additional complexity shows in the solutions presented here.

The form of the Basic Linear Process Equation (BLPE) of Section 4.1, is adapted to incorporate parameterised actions. This newly obtained form is referred to as a *Linear Process Equation* (see also Chapter 2), and extends the definition of the BLPE. Notice that it again deals with processes that can run indefinitely, or terminate unsuccessfully.

Definition 4.2.1 (*Linear Process Equation*).
A process equation is called a *Linear Process Equation* if it is of the form of Eqn. (4.9).

$$P(d{:}D) = \sum_{i \in I} \sum_{e_i : D_i} [b_i(d, e_i)] :: \!\to a_i(f_i(d, e_i)) \cdot P(g_i(d, e_i)) \tag{4.9}$$

The function f_i encodes the dependency of the parameter of an action a_i on the current state d and the values from the domain D_i. The effect of executing action a_i is coded in the function g_i. Enabledness of an action a_i with parameter $f_i(d, e_i)$ is governed by the boolean function b_i.

The effect of executing a (parameterised) uncontrollable action on the system's state can still be calculated by the *autonomous reachability* relation Ω_A, defined in Def. 4.1.3. Care must be taken, however, to cover the consequences of the parameters that accompany the execution of a controllable action. To this end, we extend the definition of the *controllable autonomous reachability* relation.

Definition 4.2.2 (*Controllable Autonomous Reachability*).
The *controllable autonomous reachability* relation $\Omega_{c(f),A}{:}2^D \to 2^D$ represents the set of states that can be reached autonomously by the system *after* executing the parameterised action $c(f)$. Since the value f that is passed along with the execution of action c needs not uniquely determine

a state in the parameter-space D, we must accept that it could have originated from more than one state in the parameter-space D, leading to a set of reachable states.

$$\begin{aligned}\Omega_{c(f),A} = \Omega_A(\{g_i(d,e_i) \in D \mid \ & i \in I \wedge a_i = c \wedge d \in \hat{d} \wedge \\ & e_i{:}D_i \wedge f = f_i(d,e_i) \wedge b_i(d,e_i)\})\end{aligned} \tag{4.10}$$

The slight alteration of the definition of the controllable autonomous reachability relation calls for a change in the definition of a ι-restrictor, since it must be able to synchronise on parameterised actions.

Definition 4.2.3 (*ι-Restrictor*).
Let $P{:}D \to \mathbb{P}$ be a Linear Process Equation with controllable action set A_c and uncontrollable action set A_u, denoting a system's behaviour. Let $\iota{:}D \to \mathbb{B}$ be a predicate on the parameter-space of P, representing a desired property of the system. The initial safe-set $S \subseteq D$ is as defined for the ι-restrictor for BLPE-processes (see Def. 4.1.8). The *ι-restrictor* $C{:}2^D \to \mathbb{P}$ for this system is defined by Eqn. (4.11)

$$C(\hat{d}{:}2^D) = \sum_{a \in A_c} \ \sum_{f:\bigcup\{f' \in \mathsf{ran}(f_i) \mid a_i = a\}} \overline{a}(f) \cdot \ C(\Omega_{a(f),A_u}(\hat{d})) \ \triangleleft \, \Omega_{a(f),A_u}(\hat{d}) \subseteq \mathcal{I} \triangleright \delta \tag{4.11}$$

Showing that the above defined ι-restrictor for processes with parameterised actions is a ι-controller requires a straightforward adaptation of the proofs of Lemma 4.1.9 and Theorem 4.1.10. Given the close correspondence, we omit the results and proofs thereof in this section. Instead, we provide two small examples, to which the theory is applied.

Example 4.2.4. Assume a simple lottery game can be played, in which a random number is drawn, which can subsequently be shown on a screen (if it is decided to do so); the screen is represented by a list of natural numbers. For simplicity, assume the system can run indefinitely. A μCRL representation of this game is found in Table 4.1. Suppose a crook has gained access

proc *Lottery*$(i{:}\mathbb{N}, l{:}[\mathbb{N}]) =$
show$(i) \cdot$ *Lottery*$(i, i \vdash l)$
$+ \sum_{j:\mathbb{N}}$ *random* $\cdot$ *Lottery*(j, l)

Table 4.1: Simple Lottery Game. Here, $[\mathbb{N}]$ stands for a list of natural numbers, $i \vdash l$ represents the list $[i] +\!\!+ l$, where $[i]$ is a list consisting of the element i and $+\!\!+$ is list concatenation

to the control panel of the lottery system, thereby obtaining the control over the numbers that are shown on a screen. Furthermore, assume the villain bought several tickets, and his objective is to win the lottery using one of the tickets he has. A safe strategy for him is dictated by the process *Crook* of Table 4.2, which is actually a ι-controller for the property $l \in \mathsf{pref}(Z)$, where Z represents the set of lottery-tickets owned by the villain. Note that there is still some redundancy in the parameters of the *Crook* process, as the crook only needs to retain information concerning the history of the numbers that have been shown up till now.

$$\textbf{proc}\ \mathit{Crook}(\hat{d}{:}2^{\mathbb{N}\times[\mathbb{N}]}) =$$

$$\textstyle\sum_{n:\mathbb{N}}\ \overline{\mathit{show}}(n)\cdot \mathit{Crook}(\{(j, n\vdash l) | (i,l)\in\hat{d}\wedge j\in\mathbb{N}\}) \lhd \forall_{(i,l)\in\hat{d}}\, n\vdash l\in \mathsf{pref}(Z) \rhd \delta$$

Table 4.2: Villain Controlling Outcome of Lottery

Example 4.2.5. A more useful example in the setting of communications protocols is to consider the example of an unreliable communications channel that has unbounded capacity. The unreliability of the channel is tightly coupled to the current number of messages in the channel (greater or smaller than a certain threshold L). A specification for this channel can be found in Table 4.3. Assume the controllable actions are *read* and *send*, whereas the uncontrollable action is *lose*. Using the threshold information about the channel, we can formulate a desired

$$\textbf{proc}\ \mathit{Channel}(q{:}[D]) =$$

$$\begin{aligned}&\textstyle\sum_{d:D}\ \mathit{read}(d)\cdot \mathit{Channel}(d\vdash q)\\ +&\textstyle\sum_{d:D,q':[D]}\ \mathit{send}(d)\cdot \mathit{Channel}(q')\lhd q = q' +\!\!+ [d]\rhd\delta\\ +&\textstyle\sum_{d:D,q',q'':[D]}\ \mathit{lose}(d)\cdot \mathit{Channel}(q' +\!\!+ q'')\lhd q = q' +\!\!+ [d] +\!\!+ q''\wedge L<|q|\rhd\delta\end{aligned}$$

Table 4.3: Unreliable Channel

property, such that we are sure that the channel always behaves reliably. Define $\iota{:}[D]\rightarrow\mathbb{B}$ as $\iota(q) = |q|\leq L$. The ι-restrictor we obtain (after simplifications) is *Limit*:$\mathbb{N}\rightarrow\mathbb{P}$, and is shown in Table 4.4.

$$\textbf{proc}\ \mathit{Limit}(n{:}\mathbb{N}) =$$

$$\begin{aligned}&\textstyle\sum_{d:D}\ \overline{\mathit{read}}(d)\cdot \mathit{Limit}(n+1)\lhd n+1\leq L\rhd\delta\\ +&\textstyle\sum_{d:D}\ \overline{\mathit{send}}(d)\cdot \mathit{Limit}(n-1)\lhd 0\leq n-1\rhd\delta\end{aligned}$$

Table 4.4: Restrictor to Guarantee Reliable use of the Unreliable Channel

4.3 Safety and Controllability

The theory developed in the previous Sections deals with restricting behaviour of processes in such a way that their executions guarantee no desired properties are violated. However, in some cases, brute force is needed (halting the system's executions) to provide these guarantees. In the next two sections, we investigate systems that do not have to resort to these methods. In Section 4.3.1 we define the notion of safety and strong safety, and in Section 4.3.2 we investigate a notion of controllability and strong controllability.

4.3.1 Safety and Strong Safety

In this section, we investigate the notions of *safety* and *controllability*. We restrict our attention to processes that can be written in the LPE format of Section 4.2. First, we define what it means for a process to be *safe* and *strongly safe* with respect to a property. Recall Def. 4.1.7, defining the notion of *ι-admissible states*. Let for a given property ι, Γ be the set of ι-admissible states.

Definition 4.3.1 (*Safe, Strongly Safe*).
Let $P{:}D \rightarrow \mathbb{P}$ be a process and $\iota{:}D \rightarrow \mathbb{B}$ be a predicate, representing a property. Process P is called *safe* with respect to property ι iff there is at least one admissible state, i.e. $\Gamma \neq \emptyset$. Process P is called *strongly safe* with respect to property ι iff process P is safe and all states satisfying property ι are admissible, i.e. $\Gamma = \mathcal{I}$.

For processes that are (strongly) safe with respect to some predicate ι, we know there exists a ι-restrictor that ensures the predicate is also an invariant of these processes. In other words, we can find parameters, satisfying the invariant, such that the restricted process does not immediately deadlock. Safety alone, however, not always suffices. In many cases, absence of deadlock, in combination with some safety objectives, is studied. If, given a process, property and restrictor, the synchronous product of the process and the restrictor (i.e. the process that is obtained by blocking individual controllable and co-controllable actions whilst allowing the result of their communication) is free of deadlock, we refer to the restrictor as a *supervisor*.

4.3.2 Controllability and Strong Controllability

The additional requirements of absence of deadlock, on top of (strong) safety is referred to as *(strong) controllability*. We start by defining what we mean by the notions *controllable* and *strongly controllable*.

Definition 4.3.2 (*Controllable, Strongly Controllable*).
Let $P{:}D \rightarrow \mathbb{P}$ be a process, $\iota{:}D \rightarrow \mathbb{B}$ be a predicate and $C{:}2^D \rightarrow \mathbb{P}$ be the ι-restrictor as defined in Def. 4.2.3. Then, process P is *controllable* with respect to ι iff process P is safe and there exists an admissible state $d \in \Gamma$, for which the synchronous product of the process $P(d)$ and the ι-restrictor $C(\Omega_{A_u}(\{d\}))$ is free of deadlock. Process P is *strongly controllable* iff process P is controllable for all admissible states.

Checking whether an initialised process is controllable with respect to a given property has the complexity of checking for deadlock of a particular instantiation of the process and its restrictor. In this chapter, our main interest is in maintaining invariance of a property for all allowed instantiations of a process' parameters. The notion of strong safety in combination with strong controllability is therefore closer to our intentions than mere safety and controllability. Dependent on the size of the admissible state set Γ, naively checking whether a process is strongly controllable can be massively complex. Its worst case complexity is dependent on the number of elements in 2^Γ.

In this section, we investigate two classes of systems for which the question of strong controllability can be answered more readily. The first class we consider is the class of processes for which the restrictor can decide to allow a controllable action by checking whether this action is allowed if the restrictor has no information about the state the process actually is in. In other words, the restrictor has exactly the information it would have when it thinks the process can be in *all* allowed states. We refer to this class of processes as *strict processes*. Notice that we here somewhat smuggle, since it is actually the combination of the process and a property that determines whether a process is strict or not.

Definition 4.3.3 (*Strict Processes*).
Let $P{:}D \rightarrow \mathbb{P}$ be a process, $\iota{:}D \rightarrow \mathbb{B}$ a predicate, $C{:}2^D \rightarrow \mathbb{P}$ be the ι-restrictor and $\Gamma \subseteq D$ be the admissible state set of process P. Process P is called *strict*, iff Eqn. (4.12) is satisfied.

$$\pi_1(\sum_{d\in\Gamma} C(\Omega_{A_u}(\{d\}))) = \pi_1(C(\Gamma)) \tag{4.12}$$

The projection function $\pi_n{:}\mathbb{P} \rightarrow \mathbb{P}$ is as defined in [18], and extended to pCRL operators, defined by Eqn. (4.13).

$$\begin{array}{lcl} \pi_1(a \cdot x) & = & a \\ \pi_n(a) & = & a \\ \pi_{n+1}(a \cdot x) & = & a \cdot \pi_n(x) \\ \pi_n(x + y) & = & \pi_n(x) + \pi_n(y) \\ \pi_n(x \lhd b \rhd y) & = & \pi_n(x) \lhd b \rhd \pi_n(y) \\ \pi_n(\sum_d x) & = & \sum_d \pi_n(x) \end{array} \tag{4.13}$$

here, a represents an atomic action or the constant δ, b is a boolean expression and x and y are process variables. In effect, by applying the function π_1, the initial actions are made explicit.

In fact, a very restricted class of processes is strict. Before arriving at sufficient conditions that enable us to establish the (strong) controllability of a strict process, we first observe that the *restricted* strict processes can in fact be rewritten to less complex processes, as the restriction can be achieved statically.

Theorem 4.3.4 (*Static Restriction*).
Let $P{:}D \rightarrow \mathbb{P}$ be a strict process and let Γ be the admissible states set. Then, process P can be statically restricted, i.e. there exists a set B of actions, such that for all admissible states d relation (4.14) holds.

$$\partial_{A_c \cup \overline{A}_c}(P(d) \| C(\Omega_{A_u}(\{d\}))) \;\leftrightarrows\; \rho_R\,(\partial_B\; P(d)) \tag{4.14}$$

Communication and renaming are defined for all $a \in A_c$ by γ and R, where $\gamma(a, \overline{a}) = \mathsf{a}$ and $R(a) = \mathsf{a}$.

Proof. Let $P{:}D \rightarrow \mathbb{P}$ be a process with controllable action set A_c, $\iota{:}D \rightarrow \mathbb{B}$ be a predicate and $C{:}2^D \rightarrow \mathbb{P}$ be the ι-restrictor. Let Γ be the admissible states set. We start by constructing the set of controllable actions B that need to be encapsulated.

$$B = \{a \in A_c | \overline{a} \not\subseteq \pi_1(C(\Gamma))\} \tag{4.15}$$

where $x \subseteq y$ is true iff $x + y = y$. Let process $PC{:}D \times 2^D \rightarrow \mathbb{P}$ be the synchronous product of the process P and the ι-restrictor C, i.e. $PC(d, \hat{d}) = \partial_{A_c \cup \overline{A}_c}(P(d) \| C(\hat{d}))$. It suffices to show there exists a bisimulation relation $\mathcal{R}$ between process $PC(d, \Omega_{A_u}(\{d\}))$ and $\rho_R(\partial_B P(d))$ for all admissible d. Let $\mathcal{R}$ be defined by Eqn. (4.16).

$$\mathcal{R} = \{(PC(d, \hat{d}), \rho_R(\partial_B P(d)))|d \in \hat{d} \wedge \hat{d} = \Omega_{A_u}(\hat{d})\} \tag{4.16}$$

We proceed by showing that $\mathcal{R}$ is a bisimulation relation, relating process $PC(d, \Omega_{A_u}(\{d\}))$ and process $\rho_R(\partial_B P(d))$. Assume $PC(d, \hat{d})$ and $\rho_R(\partial_B P(d))$ are related via $\mathcal{R}$.

- Assume process $PC(d, \hat{d}) \xrightarrow{a(f)} PC(d_0, \hat{d}_0)$. There are two cases we need to investigate.

  1. Suppose action a is an uncontrollable action. Then $\rho_R(\partial_B P(d)) \xrightarrow{a(f)} \rho_R(\partial_B P(d_0))$, and by default, processes $PC(d_0, \hat{d}_0)$ and $\rho_R(\partial_B P(d_0))$ are related via $\mathcal{R}$.
  2. Suppose action a is due to a communication between a controllable action a' and a co-controllable action $\overline{a}'$, such that $P(d) \xrightarrow{a'(f)} P(d_0)$ and $C(\hat{d}) \xrightarrow{\overline{a}'(f)} C(\hat{d}_0)$. Since process P is strict, we know that if a controllable action is permitted in a small control set $\hat{d}$, then it is also permitted in the largest possible control set Γ. Therefore, $\overline{a}' \subseteq \pi_1(C(\Gamma))$, i.e. $a' \in B$. Thus $\rho_R(\partial_B P(d)) \xrightarrow{a(f)} \rho_R(\partial_B P(d_0))$. Again, by default, processes $PC(d_0, \hat{d}_0)$ and $\rho_R(\partial_B P(d_0))$ are related via $\mathcal{R}$.

- Assume process $\rho_R(\partial_B P(d)) \xrightarrow{a(f)} \rho_R(\partial_B P(d_0))$. Again, we have two cases we need to investigate.

  1. Suppose action a is an uncontrollable action. It follows that $P(d) \xrightarrow{a(f)} P(d_0)$. Since a is uncontrollable, we also have in any context $\hat{d}$ for which $\hat{d} = \Omega_{A_u}(\hat{d})$ and $d \in \hat{d}$, that $PC(d, \hat{d}) \xrightarrow{a(f)} PC(d_0, \hat{d})$. By definition, $d_0 \in \hat{d}$, and therefore, we have that processes $\rho_R(\partial_B P(d_0))$ and $PC(d_0, \hat{d})$ are related via $\mathcal{R}$.
  2. Suppose action a is due to a renaming of a controllable action a'. Then, it follows that $P(d) \xrightarrow{a'(f)} P(d_0)$, since $a' \in A_c$ is obviously not in B. Therefore, we know $a' \subseteq \pi_1(C(\Gamma))$. Thus, we know action a' is enabled in state d and, since process P is strict, we know that action a' is enabled in any control set $\hat{d} = \Omega_{A_u}(\hat{d})$ containing state d. Therefore, also $C(\hat{d}) \xrightarrow{\overline{a}'(f)} C(\hat{d}_0)$ for $\hat{d}_0 = \Omega_{a'(f), A_u}(\hat{d})$, such that $d_0 \in \hat{d}_0$. Thus, also $PC(d, \hat{d}) \xrightarrow{\overline{a}'(f)} PC(d_0, \hat{d}_0)$. Obviously, $d_0 \in \hat{d}_0$, and therefore, by definition, we know $\rho_R(\partial_B P(d_0))$ and $PC(d_0, \hat{d}_0)$ are related via $\mathcal{R}$.

□

Proving a strict process is strongly controllable is obviously less of a challenge than for processes in general. As already remarked, one of the major complexity issues in proving a process strongly controllable is the size of the set of control states (the parameter-space) of the restrictor. Since for strict processes, the restrictor can be eliminated, the entire question of strong controllability depends on the strict process itself. In fact, if we take a closer look at the definition of strong controllability, there is the observation that a process can only deadlock in states where there is no possible uncontrollable activity. We refer to such states as *exits*. For any non-exit, the process itself can always execute non-controllable actions, but for exit states, the process must be authorised by the controller to execute a controllable action (if such an action is possible to begin with). Notice that exits do not depend on the desired property ι, but on the partitioning of the set of actions into controllable actions and uncontrollable actions.

Definition 4.3.5 (*Exits*).
Let $P{:}D \to \mathbb{P}$ be a process, $\iota{:}D \to \mathbb{B}$ be a predicate and let Γ be the set of admissible states, induced by ι. Let A_u be the uncontrollable actions of P. A state $d \in \Gamma$ is referred to as an *exit* of P iff $\Omega_{A_u}(\{d\}) = \{d\}$.

We return to the question of strong controllability for strict processes. The observation that a process can only deadlock at exits yields the following sufficient condition for proving that a strict process is strongly controllable.

Theorem 4.3.6 (*Strong Controllability of Strict Processes*).
Let $P{:}D \to \mathbb{P}$ be a strict process, let $\iota{:}D \to \mathbb{B}$ be a predicate and let Γ be the set of admissible states, induced by ι. Let $C{:}2^D \to \mathbb{P}$ be the ι-restrictor. Then, P is strongly controllable iff P is safe and for all exits $d \in \Gamma$ Eqn. (4.17) holds.

$$\pi_1(\rho_R(P(d))) \cap \pi_1(C(\{d\})) \neq \delta \tag{4.17}$$

Here, we define $\cap{:}\mathbb{P} \times \mathbb{P} \to \mathbb{P}$ as $x \cap y = \sum_{z:\mathbb{P}} z \triangleleft z \subseteq x \wedge z \subseteq y \triangleright \delta$. The mapping $R{:}A_c \to \overline{A}_c$ is defined in the obvious way as $R(a) = \overline{a}$ for all $a \in A_c$.

Proof. Let $P{:}D \to \mathbb{P}$ be a safe and strict process. Then, it is obvious that if P is strongly controllable, then process P is controllable for all admissible states $d \in \Gamma$, hence, P is also controllable for a subset of Γ.
Suppose for all exits $d \in \Gamma$, we have $\pi_1(\rho_R(P(d))) \cap \pi_1(C(\{d\})) \neq \delta$. Let d_0 be an admissible state and suppose process $P(e)$ is reachable from process $P(d_0)$ and deadlocks. This means, process $P(e)$ cannot perform an uncontrollable action, i.e. state e must be an exit. However, according to our assumption, exits have at least one enabled controllable action. Thus, process $P(d)$ cannot deadlock and therefore process P is strongly controllable. $\square$

In fact, for general processes, we can also formulate a constraint on the allowed behaviour at the exits of these processes, such that we can safely conclude strong controllability. The requirement, however, is still rather severe, but the class of strongly controllable processes that is identified here is larger than the class of strict processes.

Theorem 4.3.7 (*Strong Controllability of General Processes*).
Let $P{:}D \to \mathbb{P}$ be an arbitrary safe process, let $\iota{:}D \to \mathbb{B}$ be a predicate and let $\Gamma \subseteq D$ be the set of admissible states, induced by ι. Let $C{:}2^D \to \mathbb{P}$ be the ι-restrictor. Then, process P is strongly controllable iff for all exits d of P, Eqn. (4.18) holds.

$$\pi_1(\rho_R P(d)) \cap \pi_1(C(\Gamma)) \neq \delta \tag{4.18}$$

where the mapping $R{:}A_c \to \overline{A}_c$ is defined as $R(a) = \overline{a}$ for all $a \in A_c$.

Proof. Let process $P{:}D \to \mathbb{P}$ be a safe process, for which all exits d satisfy $\pi_1(\rho_R P(d)) \cap \pi_1(C(\Gamma)) \neq \delta$. We proceed by showing that process $PC(d, \hat{d}) = \partial_{A_c \cup \overline{A}_c}(P(d) \| C(\hat{d}))$ cannot deadlock for arbitrary initial state $d' \in \Gamma$.
Assume that after a number of steps (possibly zero), process $PC(d', \Omega_{A_u}(\{d'\}))$ reaches process $PC(e_0, \hat{e}_0)$ and deadlocks. There are two possible scenarios; either state e_0 is an exit, or state e_0 is not an exit.

- Suppose state e_0 is not an exit. Then, this means process P can perform an uncontrollable action, not requiring the cooperation of process C and therefore process $PC(e_0, \hat{e}_0)$ cannot deadlock.
- Suppose state e_0 is an exit. According to our assumption, we now know $\pi_1(\rho_R P(e_0)) \cap \pi_1(C(\Gamma)) \neq \delta$. This means $\pi_1(P(e_0)) \neq \delta$. In fact, there is an action $\overline{a} \subseteq \pi_1(\rho_R P(e_0)) \cap \pi_1(C(\Gamma))$, and, since $e_0 \in \hat{e}_0$, also $\overline{a} \in \pi_1(\rho_R P(e_0)) \cap \pi_1(C(\hat{e}_0))$. Thus, we know an action is enabled, so process $PC(e_0, \hat{e}_0)$ cannot deadlock.

□

The above conditions for proving strong controllability for processes (either strict processes or more general processes) are based on the intuition that for every exit a process can perform at least one controllable action that is not disallowed in any context. In a way, this means that some or all local decisions can always be resolved at a global level.

In the remainder of this section, we investigate a more liberal class of processes. The complexity of proving strong controllability of processes that are part of this class is reduced by employing the structure of the parameter-space of the restrictor. To this end, the concept of an *environment* is introduced.

Definition 4.3.8 (*Environment*).
Let $P{:}D \to \mathbb{P}$ be a safe process with uncontrollable action set A_u and let $\Gamma \subseteq D$ be the set of admissible states. An *environment* Π_d of an admissible state $d \in \Gamma$ is a subset of Γ, defined by Eqn. (4.19).

$$\Pi_d = \bigcup_{d' \in \Gamma} \{\Omega_{A_u}(\{d'\}) \mid \Omega_{A_u}(\{d'\}) \cap \Omega_{A_u}(\{d\}) \neq \emptyset\} \tag{4.19}$$

An environment is a cluster of admissible states such that all states inside this environment are linked via uncontrollable actions only. Environments satisfy some intuitive properties.

Property 4.3.9 Let $P{:}D \to \mathbb{P}$ be a safe process with uncontrollable action set A_u and let $\Gamma \subseteq D$ be the set of admissible states. Then, for all $d, e \in \Gamma$, we have

- If $\Pi_d \cap \Pi_e = \emptyset$, then $d \neq e$,
- $d \in \Pi_d$,
- $\Pi_d = \Pi_e$ iff $\Pi_d \cap \Pi_e \neq \emptyset$,
- $\bigcup_{d \in \Gamma} \Pi_d = \Gamma$,
- $\Omega_{A_u}(\Pi_d) = \Pi_d$.

Proof. The proof follows immediately from Property 4.1.4. □

The environments are artificial structures on the set of admissible states. However, we can use this structure to characterise a class of processes that are less exotic in their behaviour. In general, one of the major hurdles in proving a process is strongly controllable is the difficulty in bringing order into the states (control set) the restrictor thinks a process can be in.

If, however, this control set always is (part of) a uniquely identifiable environment, the proof suddenly gets less complex. An obvious structure to study is based on *determinism*. Determinism is based on atomic actions reaching individual states. Rather than taking these individual states, we study a notion of determinism using environments. This notion of determinism is called *meta-determinism*.

Definition 4.3.10 (*Meta-Determinism*).
Let $P{:}D \to \mathbb{P}$ be a process and $\iota{:}D \to \mathbb{P}$ be a desired property. Let $E \subseteq \Gamma$ be a smallest set such that $\bigcup_{d \in E} \Pi_d = \Gamma$. Then, process P is *meta-deterministic* iff condition (4.20) is satisfied.

$$\forall_{e \in E, a \in A_c, f \in \bigcup\{f' \in \mathsf{ran}(f_i) | a_i = a\}} \exists_{e' \in E} \forall_{d \in \Pi_e} \Omega_{a(f),A_u}(\{d\}) \subseteq \mathcal{I} \Rightarrow \Omega_{a(f),A_u}(\{d\}) \subseteq \Pi_{e'} \quad (4.20)$$

Lemma 4.3.11.
Let $P{:}D \to \mathbb{P}$ be a meta-deterministic process, $\iota{:}D \to \mathbb{B}$ be a predicate, and $C{:}2^D \to \mathbb{P}$ be the ι-restrictor. Define $PC(d, \hat{d}) = \partial_{A_c \cup \overline{A}_c}(P(d) \| C(\hat{d}))$. Then, process $PC{:}D \times 2^D \to \mathbb{P}$ satisfies the invariant $\hat{d} \subseteq \Pi_d$.

Proof. Let process $PC{:}D \times 2^D \to \mathbb{P}$ be as defined by Lemma 4.3.11. Assume $\hat{d} \subseteq \Pi_d$ and assume process PC executes an action a. We distinguish the two types of actions, viz. controllable and uncontrollable actions.

1. Suppose action a is due to the communication of a controllable action and a co-controllable action. Suppose we have a transition $PC(d, \hat{d}) \xrightarrow{a(f)} PC(g_i(d, e_i), \Omega_{a(f),A_u}(\hat{d}))$, for given i and e_i. Since process P is meta-deterministic, we know there is an environment e, such that for all $d' \in \Pi_d$, we have $\Omega_{a(f),A_u}(\{d'\}) \subseteq \Pi_e$, and therefore also $\bigcup_{d' \in \Pi_d} \Omega_{a(f),A_u}(\{d'\}) \subseteq \Pi_e$. Since we start from the assumption that $\hat{d} \subseteq \Pi_d$, we also know $\bigcup_{d' \in \hat{d}} \Omega_{a(f),A_u}(\{d'\}) \subseteq \bigcup_{d' \in \Pi_d} \Omega_{a(f),A_u}(\{d'\}) \subseteq \Pi_e$. From Property 4.1.4 we can derive that also $\Omega_{a(f),A_u}(\hat{d}) \subseteq \Pi_e$. From Lemma 4.1.9, it immediately follows that $g_i(d, e_i) \in \Pi_e$. Property 4.3.9 then allows us to derive that $\Omega_{a(f),A_u}(\hat{d}) \subseteq \Pi_{g_i(d,e_i)}$.

2. Suppose action a is due to an uncontrollable action of process P. Suppose we have a transition $PC(d, \hat{d}) \xrightarrow{a(f)} PC(g_i(d, e_i), \hat{d})$, for given i and e_i. Because of Lemma 4.1.9, we have $g_i(d, e_i) \in \hat{d}$. We work under the assumption that $\hat{d} \subseteq \Pi_d$. Therefore, also $g_i(d, e_i) \in \Pi_d$, and from Property 4.3.9 we can deduce that $\Pi_d = \Pi_{g_i(d,e_i)}$. Thus, also $\hat{d} \subseteq \Pi_{g_i(d,e_i)}$.

□

The above Lemma justifies the argument that the notion of meta-determinism assures the restrictor has an accurate picture of the current state of a process. A meta-deterministic process behaves like a deterministic process if we consider the environments as atomic states (however, not all deterministic processes are also meta-deterministic processes). Note that this also does not imply the process itself is necessarily deterministic, as there can be some non-deterministic behaviour that takes the process to only one single environment, yet two different states in this environment. However, the class of meta-deterministic processes does not contain non-deterministic processes that have no uncontrollable actions. In Fig. 4.1, (an abstraction of) a meta-deterministic process is portrayed. To prove a meta-deterministic process is strongly controllable, we can adapt

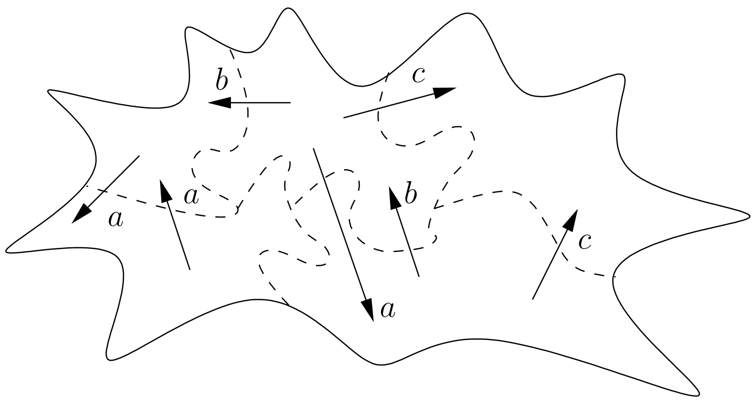


Figure 4.1: Environments and Meta-Deterministic Processes

Theorem 4.3.7 such that it is slightly more liberal. The basic idea is that for meta-deterministic processes, the restrictor always knows in which environment the process resides. Then, deadlocks only occur when an exit of the environment has no allowed action that can be executed.

Theorem 4.3.12 (*Strong Controllability and Meta-Deterministic Processes*).
Let $P{:}D \to \mathbb{P}$ be a safe and meta-deterministic process, then process P is strongly controllable iff for all environments Π_e and all exits $d \in \Pi_e$, the Eqn. (4.21) holds.

$$\pi_1(\rho_R P(d)) \cap \pi_1(C(\Pi_e)) \neq \delta \tag{4.21}$$

where $R{:}A_c \to \overline{A}_c$ is again defined in the obvious way, i.e. $R(a) = \overline{a}$ for all $a \in A_c$.

Proof. Assume $P{:}D \to \mathbb{P}$ is a meta-deterministic process, such that for all environments Π_e and all exits d, $\pi_1(\rho_R P(d)) \cap \pi_1(C(\Pi_e)) \neq \delta$. Let process $PC(d, \hat{d}) = \partial_{A_c \cup \overline{A}_c}(P(d) \| C(\hat{d}))$. We proceed by showing that process PC cannot deadlock for arbitrary initial state $d' \in \Gamma$.
Assume that after a number of steps (possibly zero), process $PC(d', \Omega_{A_u}(\{d'\}))$ reaches process $PC(e_0, \hat{e}_0)$ and deadlocks. There are two possible scenarios; either state e_0 is an exit, or state e_0 is not an exit.

1. Suppose state e_0 is not an exit. Then, process $P(e_0)$ can perform an uncontrollable action, and hence process $PC(e_0, \hat{e}_0)$ cannot deadlock.

2. Suppose state e_0 is an exit. According to our assumption, we now know $\pi_1(\rho_R P(e_0)) \cap \pi_1(C(\Pi_{e_0})) \neq \delta$. This means that $\pi_1(\rho_R P(e_0)) \neq \delta$ and therefore, also $P(e_0) \neq \delta$. In fact, there is an action $\overline{a} \subseteq \pi_1(\rho_R P(e_0)) \cap \pi_1(C(\Pi_{e_0}))$. By Lemmas 4.1.9 and 4.3.11, we know $e_0 \in \hat{e}_0$ and $\hat{e}_0 \subseteq \Pi_{e_0}$. Therefore, also $\overline{a} \subseteq \pi_1(\rho_R P(e_0)) \cap \pi_1(C(\hat{e}_0))$. Thus, we know that process $C(\hat{e}_0)$ can execute an action that is co-controllable for process $P(e_0)$, so process $PC(e_0, \hat{e}_0)$ cannot deadlock.

Thus, for both cases we can prove the process PC cannot reach a deadlock situation. □

Given a set of admissible states Γ (induced by some property ι, expressed over a process P), the number of different environments is, for a fixed set of A_u and A_c, at most $|\Gamma|$. This is easy to see: the smallest environment is a singleton environment, and therefore, the partition of Γ, consisting of singleton environments only, is of size $|\Gamma|$.

4.4 Case-study

In this section, we apply the theory of the preceding sections to a case-study, described in [26]. The case study deals with a turntable, which is a simple machine that can perform various actions. In particular, products can be removed from and placed on the turntable machine by means of resp. a *remove* and an *add* action, both of which are controllable. Added products are detected by means of a sensor reporting a controllable action tt_1. The completion of a removal of a product is detected by a sensor, reporting a controllable action tt_0. Rotating the turntable is initiated by the controllable action *turn*. The completion of the 90^o rotation is registered by a sensor reporting a controllable action tt_2. The process *TT* registers whether there are products on the positions one through four, using parameters p_0, p_1, p_2 and p_3. The $turn_{TT}$ parameter registers whether the table is turning, add_{TT} registers whether a new product is being added and rem_{TT} registers whether a product is being removed. Note that for booleans b, we abbreviate conditions $b = \mathsf{tt}$ to b. The

proc $TT(p_0, p_1, p_2, p_3, turn_{TT}, add_{TT}, rem_{TT}{:}\mathbb{B}) =$

$$\begin{aligned}
&\;\; turn \cdot TT(p_0, p_1, p_2, p_3, \mathsf{tt}, add_{TT}, rem_{TT}) \lhd turn_{TT} = \mathsf{ff} \rhd \delta \\
&+ tt_2 \cdot TT(p_3, p_0, p_1, p_2, \mathsf{ff}, add_{TT}, rem_{TT}) \lhd turn_{TT} \rhd \delta \\
&+ add \cdot TT(p_0, p_1, p_2, p_3, turn_{TT}, \mathsf{tt}, rem_{TT}) \lhd add_{TT} = \mathsf{ff} \rhd \delta \\
&+ tt_1 \cdot TT(\mathsf{tt}, p_1, p_2, p_3, turn_{TT}, \mathsf{ff}, rem_{TT}) \lhd add_{TT} \land p_0 = \mathsf{ff} \rhd \delta \\
&+ remove(p_3) \cdot TT(p_0, p_1, p_2, p_3, turn_{TT}, add_{TT}, \mathsf{tt}) \lhd rem_{TT} = \mathsf{ff} \rhd \delta \\
&+ tt_0 \cdot TT(p_0, p_1, p_2, \mathsf{ff}, turn_{TT}, add_{TT}, \mathsf{ff}) \lhd rem_{TT} \rhd \delta
\end{aligned}$$

products to be drilled have to be kept in place by means of a clamp. The clamp is controlled by means of a controllable action $clamp_{on/off}$, which either locks or unlocks the clamp. A controllable sensor c_2 denotes that the product is locked and a controllable sensor c_1 senses that the product is unlocked again. The parameter act_C of process C registers whether the clamp is active and the parameter $lock_C$ registers whether the lock is active. An important part of the turntable

proc $C(act_C, lock_C{:}\mathbb{B}) =$

$$\begin{aligned}
&\;\; clamp_{on/off} \cdot C(\neg act_C, lock_C) \lhd act_C = lock_C \rhd \delta \\
&+ c_2 \cdot C(act_C, \mathsf{tt}) \lhd act_C \land lock_C = \mathsf{ff} \rhd \delta \\
&+ c_1 \cdot C(act_C, \mathsf{ff}) \lhd act_C = \mathsf{ff} \land lock_C \rhd \delta
\end{aligned}$$

system consists of the drill. The drill can be operated in several ways. Switching the drill on or off can be controlled by means of the controllable action $\mathit{drill}_{\mathit{on/off}}$. Applying the drill to a product and removing it from the product is done by means of the controllable action $\mathit{drill}_{\mathit{up/down}}$. The controllable sensor d_2 reports when the drill has reached its down position, whereas the controllable sensor d_1 reports when the drill has reached its up position. The parameter act_D registers whether the drill is active or not, whereas the parameters mov_D and pos_D registers whether the drill is moving down resp. whether the drill has reached its down position. Finally, drilled products can be flawed. To this end, the products must be tested. The testing device is switched on by means of the controllable action $\mathit{tester}_{\mathit{up/down}}$. The outcome of the test is determined by the uncontrollable actions *good* and *poor*. The outcome is communicated to the environment via an uncontrollable action t_2. The controllable sensor t_1 denotes the tester has been switched off successfully. The tester registers the activity of the tester in parameter act_T; the parameter t_T registers whether the product has been tested and the parameter t_T registers whether the test results have been reported to the environment or not. The overall conditions that must be ensured are the following:

$$
\begin{array}{l}
\textbf{proc } D(\mathit{act}_D, \mathit{mov}_D, \mathit{pos}_D{:}\mathbb{B}) = \\
\\
\quad \mathit{drill}_{\mathit{on/off}} \cdot D(\neg \mathit{act}_D, \mathit{mov}_D, \mathit{pos}_D) \\
+\mathit{drill}_{\mathit{up/down}} \cdot D(\mathit{act}_D, \neg \mathit{mov}_D, \mathit{pos}_D) \triangleleft \mathit{mov}_D = \mathit{pos}_D \triangleright \delta \\
+d_2 \cdot D(\mathit{act}_D, \mathit{mov}_D, \mathsf{tt}) \triangleleft \mathit{mov}_D \wedge \mathit{pos}_D = \mathsf{ff} \triangleright \delta \\
+d_1 \cdot D(\mathit{act}_D, \mathit{mov}_D, \mathsf{ff}) \triangleleft \mathit{mov}_D = \mathsf{ff} \wedge \mathit{pos}_D \triangleright \delta
\end{array}
$$

$$
\begin{array}{l}
\textbf{proc } T(\mathit{act}_T, t_T, r{:}\mathbb{B}) \\
\\
\quad \mathit{tester}_{\mathit{up/down}} \cdot T(\neg \mathit{act}_T, t_T, r) \triangleleft \mathit{act}_T = r \triangleright \delta \\
+\mathit{good} \cdot T(\mathit{act}_T, \mathsf{tt}, r) \triangleleft \mathit{act}_T \wedge r = \mathsf{ff} \wedge t_T = \mathsf{ff} \triangleright \delta \\
+\mathit{poor} \cdot T(\mathit{act}_T, \mathsf{tt}, r) \triangleleft \mathit{act}_T \wedge r = \mathsf{ff} \wedge t_T = \mathsf{ff} \triangleright \delta \\
+t_2 \cdot T(\mathit{act}_T, \mathsf{ff}, \mathsf{tt}) \triangleleft \mathit{act}_T \wedge r = \mathsf{ff} \wedge t_T \triangleright \delta \\
+t_1 \cdot T(\mathit{act}_T, t_T, \mathsf{ff}) \triangleleft \mathit{act}_T = \mathsf{ff} \wedge r \triangleright \delta
\end{array}
$$

1. No operations on products are permitted while the turntable is turning
   $(\mathit{turn}_{TT} \Rightarrow \neg(\mathit{act}_C \vee \mathit{lock}_C \vee \mathit{add}_{TT} \vee \mathit{rem}_{TT} \vee \mathit{act}_D \vee \mathit{mov}_D \vee \mathit{pos}_D \vee \mathit{act}_T \vee r))$,

2. Products can only be added to the table if the first slot is empty
   $(\mathit{add}_{TT} \Rightarrow \neg p_0)$

3. No turning is allowed while there is a product ready to be removed from the table
   $(\mathit{turn}_{TT} \Rightarrow \neg p_3)$,

4. Only products that are actually there can be removed
   $(\mathit{rem}_{TT} \Rightarrow p_3)$,

5. The clamp may only be used when there is a product in the right position ($\mathit{act}_C \Rightarrow p_1$ and $\mathit{lock}_C \Rightarrow p_1$)

6. The drill may only be used when there is a product in the right position ($\mathit{act}_D \Rightarrow p_1$)

7. The drill must be switched on before and during drilling the product ($\mathit{mov}_D \Rightarrow \mathit{act}_D$ and $\mathit{pos}_D \Rightarrow \mathit{act}_D$),

8. Before drilling, products must be locked ($\mathit{act}_D \Rightarrow (\mathit{lock}_C \wedge \mathit{act}_C)$ and $\mathit{mov}_D \Rightarrow (\mathit{lock}_C \wedge \mathit{act}_C)$ and $\mathit{pos}_D \Rightarrow (\mathit{lock}_C \wedge \mathit{act}_C)$),

9. The tester can only be used if there is a product to be tested ($\mathit{act}_T \Rightarrow p_2$ and $r \Rightarrow p_2$)

Define the collection of all requirements as our desired property ι. Following the recipe of Def. 4.1.8 for constructing a ι-restrictor, we arrive at a process $\mathit{MC}{:}2^{\mathbb{B}^{15}} \rightarrow \mathbb{P}$. Based on calculations using the axiom system of μCRL, we can further reduce this process to the process $\mathit{MC}{:}\mathbb{B}^{14} \rightarrow \mathbb{P}$, shown below. Using a brute-force approach we can calculate the number of states that satisfy the property ι to be 826 out of the total number of states 2^{15}. This means that by enforcing the property ι, it is sufficient to reason about approx. $100 * \frac{826}{2^{15}} \approx 2.5\%$ of the state-space. The property space, consisting of 826 states, is uniquely partitioned in 583 environments of size 1 and 81 environments of size 3.

Given that the turntable model is deterministic, we know that each environment of size 1 satisfies the meta-determinism property; for environments of size 3, the verification of the meta-determinism property is slightly more elaborate, and is not further explained.

Since each environment of size 3 has a single exit, it suffices to check whether from this exit, always a controllable action is possible. It turns out that action $\mathit{tester}_{\mathit{up/down}}$ is in these cases always enabled; for singular environments, it can be verified that in each case at least either the turntable is starting or ending a turn, a product is added or a product is removed. Using Theorem 4.3.12, the turntable is therefore proved strongly controllable with respect to the property ι.

Using additional verification, we can establish that the restricted process still allows for the turning of the table, and the adding, removing, drilling and testing of products. Thus, several intuitive liveness requirements are not in conflict with the (enforced) safety requirements.

4.5 Closing Remarks

Related Work The problem we considered in this chapter is a single-step control problem for uninitialised systems. The single-step control problem has received relatively little attention, the only noteworthy papers being [3, 4]. Whereas we study the problem in the setting of a process algebra with data, in [3, 4] the problem is studied in a setting of statically and dynamically typed *modules*, and mainly focuses on the complexity and decidability of the problem in a purely synchronous setting. Obviously, this is different from what we studied in this chapter.

The multi-step control problem, on the other hand, is studied more extensively. This problem can be seen as a generalisation of the single-step problem, and its solution is often found by iteratively solving the single-step solution. Early accounts are by Ramadge and Wonham [106, 109],

$$\begin{array}{l}
\textbf{proc } \mathit{MC}(p_0, p_1, p_2, p_3, \mathit{turn}_{TT}, \mathit{add}_{TT}, \mathit{rem}_{TT}, \mathit{act}_C, \mathit{lock}_C, \mathit{act}_D, \mathit{mov}_D, \mathit{pos}_D, \mathit{act}_T, r{:}\mathbb{B}) = \\
\\
\quad \overline{\mathit{turn}} \cdot MC(p_0, p_1, p_2, p_3, \mathsf{tt}, \mathit{add}_{TT}, \mathit{rem}_{TT}, \mathit{act}_C, \mathit{lock}_C, \mathit{act}_D, \mathit{mov}_D, \mathit{pos}_D, \mathit{act}_T, r) \\
\qquad \lhd \neg(\mathit{act}_T \vee \mathit{act}_D \vee \mathit{act}_C \vee \mathit{lock}_C \vee \mathit{add}_{TT} \vee \mathit{rem}_{TT} \vee p_3 \vee \mathit{pos}_D \vee \mathit{mov}_D \vee r) \rhd \delta \\
+\overline{\mathit{add}} \cdot MC(p_0, p_1, p_2, p_3, \mathit{turn}_{TT}, \mathsf{tt}, \mathit{rem}_{TT}, \mathit{act}_C, \mathit{lock}_C, \mathit{act}_D, \mathit{mov}_D, \mathit{pos}_D, \mathit{act}_T, r) \\
\qquad \lhd \neg \mathit{turn}_{TT} \wedge \neg p_0 \rhd \delta \\
+\overline{\mathit{rem}} \cdot MC(p_0, p_1, p_2, p_3, \mathit{turn}_{TT}, \mathit{add}_{TT}, \mathsf{tt}, \mathit{act}_C, \mathit{lock}_C, \mathit{act}_D, \mathit{mov}_D, \mathit{pos}_D, \mathit{act}_T, r) \\
\qquad \lhd \neg \mathit{turn}_{TT}) \wedge p_3 \rhd \delta \\
+\overline{\mathit{clamp}}_{\mathit{on/off}} \cdot MC(p_0, p_1, p_2, p_3, \mathit{turn}_{TT}, \mathit{add}_{TT}, \mathit{rem}_{TT}, \neg \mathit{act}_C, \mathit{lock}_C, \mathit{act}_D, \mathit{mov}_D, \mathit{pos}_D, \mathit{act}_T, r) \\
\qquad \lhd (\neg \mathit{act}_C \Rightarrow (\mathit{turn}_{TT} \wedge p_1)) \wedge ((\mathit{act}_D \vee \mathit{mov}_D \vee \mathit{pos}_D) \Rightarrow (\mathit{lock}_C \wedge \neg \mathit{act}_C)) \rhd \delta \\
+\overline{\mathit{drill}}_{\mathit{on/off}} \cdot MC(p_0, p_1, p_2, p_3, \mathit{turn}_{TT}, \mathit{add}_{TT}, \mathit{rem}_{TT}, \mathit{act}_C, \mathit{lock}_C, \neg \mathit{act}_D, \mathit{mov}_D, \mathit{pos}_D, \mathit{act}_T, r) \\
\qquad \lhd \neg \mathit{act}_D \Rightarrow ((\mathit{lock}_C \vee \mathit{act}_C) \wedge \neg \mathit{turn}_{TT} \wedge p_1) \wedge ((\mathit{mov}_D \vee \mathit{pos}_D) \Rightarrow \neg \mathit{act}_D) \rhd \delta \\
+\overline{\mathit{drill}}_{\mathit{up/down}} \cdot MC(p_0, p_1, p_2, p_3, \mathit{turn}_{TT}, \mathit{add}_{TT}, \mathit{rem}_{TT}, \mathit{act}_C, \mathit{lock}_C, \mathit{act}_D, \neg \mathit{mov}_D, \mathit{pos}_D, \mathit{act}_T, r) \\
\qquad \lhd \neg \mathit{mov}_D \Rightarrow (\neg \mathit{turn}_{TT} \wedge \mathit{act}_D \wedge \mathit{lock}_C \wedge \mathit{act}_C) \rhd \delta \\
+\overline{\mathit{tester}}_{\mathit{up/down}} \cdot MC(p_0, p_1, p_2, p_3, \mathit{turn}_{TT}, \mathit{add}_{TT}, \mathit{rem}_{TT}, \mathit{act}_C, \mathit{lock}_C, \mathit{act}_D, \mathit{mov}_D, \mathit{pos}_D, \neg \mathit{act}_T, r) \\
\qquad \lhd \neg \mathit{act}_T \Rightarrow (\neg \mathit{turn}_{TT} \wedge p_2) \rhd \delta \\
+\overline{\mathit{tt}}_0 \cdot MC(p_0, p_1, p_2, \mathsf{ff}, \mathit{turn}_{TT}, \mathit{add}_{TT}, \mathsf{ff}, \mathit{act}_C, \mathit{lock}_C, \mathit{act}_D, \mathit{mov}_D, \mathit{pos}_D, \mathit{act}_T, r) \\
+\overline{\mathit{tt}}_1 \cdot MC(\mathsf{tt}, p_1, p_2, p_3, \mathit{turn}_{TT}, \mathsf{ff}, \mathit{rem}_{TT}, \mathit{act}_C, \mathit{lock}_C, \mathit{act}_D, \mathit{mov}_D, \mathit{pos}_D, \mathit{act}_T, r) \\
+\overline{\mathit{tt}}_2 \cdot MC(p_3, p_0, p_1, p_2, \mathsf{ff}, \mathit{add}_{TT}, \mathit{rem}_{TT}, \mathit{act}_C, \mathit{lock}_C, \mathit{act}_D, \mathit{mov}_D, \mathit{pos}_D, \mathit{act}_T, r) \\
+\overline{c}_1 \cdot MC(p_0, p_1, p_2, p_3, \mathit{turn}_{TT}, \mathit{add}_{TT}, \mathit{rem}_{TT}, \mathit{act}_C, \mathsf{ff}, \mathit{act}_D, \mathit{mov}_D, \mathit{pos}_D, \mathit{act}_T, r) \\
\qquad \lhd \neg(\mathit{turn}_{TT} \vee \mathit{act}_D \vee \mathit{mov}_D \vee \mathit{pos}_D) \rhd \delta \\
+\overline{c}_2 \cdot MC(p_0, p_1, p_2, p_3, \mathit{turn}_{TT}, \mathit{add}_{TT}, \mathit{rem}_{TT}, \mathit{act}_C, \mathsf{tt}, \mathit{act}_D, \mathit{mov}_D, \mathit{pos}_D, \mathit{act}_T, r) \\
+\overline{d}_1 \cdot MC(p_0, p_1, p_2, p_3, \mathit{turn}_{TT}, \mathit{add}_{TT}, \mathit{rem}_{TT}, \mathit{act}_C, \mathit{lock}_C, \mathit{act}_D, \mathit{mov}_D, \mathsf{ff}, \mathit{act}_T, r) \\
+\overline{d}_2 \cdot MC(p_0, p_1, p_2, p_3, \mathit{turn}_{TT}, \mathit{add}_{TT}, \mathit{rem}_{TT}, \mathit{act}_C, \mathit{lock}_C, \mathit{act}_D, \mathit{mov}_D, \mathsf{tt}, \mathit{act}_T, r) \\
\qquad \lhd \mathit{act}_D \rhd \delta \\
+\overline{t}_1 \cdot MC(p_0, p_1, p_2, p_3, \mathit{turn}_{TT}, \mathit{add}_{TT}, \mathit{rem}_{TT}, \mathit{act}_C, \mathit{lock}_C, \mathit{act}_D, \mathit{mov}_D, \mathit{pos}_D, \mathit{act}_T, \mathsf{tt})
\end{array}$$

Table 4.5: Restrictor for the Turntable Model

who based their work on *discrete event systems*. In their work, they restrict to deterministic processes (often called *plants*) that start from a single initial state and produce finite words (*strings*) in some alphabet. Their aim is to find a controller that restricts a plant in such a way that it produces a subset of the entire language of the plant. Their approach differs very much from the approach followed in this chapter, as it is based on language equivalence and finite words; we work in a bisimulation setting with non-deterministic, infinite process behaviours and multiple initial states.

In [76], Kupferman *et al* study the controller synthesis problem in a setting of the branching temporal logics CTL and CTL*. Unlike our processes, the plants the authors study are restricted to having a finite number of states and a single initial state. In their paper, the authors prove that the supervisory control problems for CTL and CTL* are 2EXPTIME-hard and 3EXPTIME-hard in the size of the specification respectively. Their results hold in a setting with complete information, i.e. the plant (resp., its environment) is able to observe all the signals generated by the environment (resp., the plant). They do not study the synthesis problem in a setting with incomplete information, but research suggests that the effects of including incomplete information can range from having no impact [77] to causing undecidability [102]. In [78], Kupferman and Vardi study a similar problem for the μ-calculus in the setting with incomplete information.

Finally, we mention work by Madhusudan and Thiagarajan [89]. They consider the controller synthesis problem in a distributed setting, where a process at a given *site* can evolve asynchronously with respect to processes at other sites. The processes they consider are again deterministic and have a single initial state, which is different from our setting. The authors show that three conditions (one on the processes and two on the controllers) are necessary for the decidability of the distributed controller synthesis problem in an arbitrary *architecture* (i.e. the topology of the communication network between the sites). Since we work in a setting where we have a *global* process, our results are difficult to compare with the results obtained by Madhusudan and Thiagarajan. It is likely that some of our results for the single-step control problem can be adapted to a distributed setting.

Summary In this chapter, we discussed a simple way for restricting the behaviour of uninitialised processes by turning a desired property into an invariant of the process. For this, we use a single-step controller, guaranteeing safety by inspecting the effects of taking a single transition. An obvious extension of the techniques, developed in this chapter, is the multi-step controller. Multi-step controllers have already been studied in the context of initialised systems. For such systems, synthesising multi-step controllers often have a greater complexity than synthesising single-step controllers. Most likely, this also holds for uninitialised systems.

The invariants we obtain by restricting a process' behaviour can be interpreted as the *safety requirements* of the process. Designing a controller by hand to enforce such requirements is quite challenging. Therefore, the techniques, described in this chapter are quite useful in the design trajectory. Of course, the safety requirements are only part of the set of desired properties of a system. The first approximation of the controller, obtained by process-behaviour restriction can be helpful in the further design of the controller. Moreover, it is useful to spot potential conflicts between safety requirements and liveness requirements.

Since we are often interested in the non-blocking behaviour of (uninitialised) processes, we further investigated two types of processes for which we can establish the absence of deadlock more efficiently than for arbitrary (uninitialised) processes. We find that for certain systems it suffices to block a subset of all controllable actions. For a second class of systems, we found that the controller based its decisions on single environments of a process. For both classes of processes, we found simple sufficient conditions for guaranteeing absence of deadlock in the restricted process.

The results, described in this chapter were put to use in a small case study, describing a turntable system (see Section 4.4). In this case-study, we show that using our techniques, we end up with a first approximation of the turntable system that contains only 2.5% of the number of states of the uncontrolled turntable system.

Chapter 5

Timed and Hybrid Automata

With the seminal publication [7, 8], the field of real-time process theories had finally reached maturity. The popular description technique of *Timed Automata*, first introduced in [7], allows for the description of systems in which the interplay between discrete events and the passage of time plays a pivotal role for the system's correctness. Even though a multitude of existing description techniques had already been equipped with real-time extensions, it was not until the introduction of timed automata that a full-grown analysis technique for real-time systems became available. Extensions of timed automata, capable of describing combinations between discrete events and even more exotic (from a computer scientist's point of view) behaviours, followed suit. One such extensions was dubbed *Hybrid Automata* [6, 62], which allowed to combine discrete and continuous behaviours, in other words, *hybrid systems*. Various results pertaining to the decidability of automated verification of (classes of) such systems have been obtained in the years following.

From a practical point of view, the frameworks of timed and hybrid automata are of great importance; they provide a pragmatic way that allows us to investigate complex systems. The results obtained by such investigations are often of utmost importance. On the other hand, neither timed, nor hybrid automata have thus far provided deeper insight into the more fundamental questions concerning the rôle and essence of time in real-time and hybrid systems. Such theoretical issues are the key issue in real-time process algebras.

Chapters 5 and 6 define a bridge between the more practically oriented frameworks of timed and hybrid automata and the theoretically oriented framework of μCRL$_t$. We do so, by providing a sound interpretation of timed and (a class of) hybrid automata in μCRL$_t$, and we address questions of expressiveness in all frameworks. Some of the results are traced back to the characteristics of time that can differ, dependent on the framework we work in. This chapter serves as a bridge between timed and hybrid automata on the one hand, and μCRL$_t$ on the other hand. It does so by investigating two different ways of providing a semantics for timed and hybrid automata and it clarifies conceptual differences and conformity. An obvious difference between timed and hybrid automata on the one hand and μCRL$_t$ on the other hand is the different notion of time that is employed: timed and hybrid automata use *relative* time, whereas μCRL$_t$ is based on *absolute* time. We choose the path of least resistance and embed the relative time setting into an absolute time setting. Other possibilities are well possible, but most likely require more effort.

5.1 Timed Automata

In section 5.1.1, we introduce the syntax of timed automata; section 5.1.2 subsequently formulates an absolute time version of the standard semantics of a timed automaton, whereas section 5.1.3 introduces an alternative to this standard semantics. The relation between both semantics is investigated in section 5.1.4.

5.1.1 Syntax

Timed Automata were first introduced by Alur and Dill in [7, 8]. They provided a simple and general way for annotating state-transition graphs with timing constraints using finitely many real-valued *clock variables*. Their automata, however, did not have the means to enforce progress. In the course of years, various extensions of timed automata have been defined. Noteworthy extensions introduce *deadlines* (see [25]) and invariants (see e.g. [5]). This latter extension seems to have become the *de facto* standard, which also is the one we investigate in this thesis.

In a timed automaton, timing constraints are typically modelled using *clocks*. The timing constraints that can be expressed are restricted to simple predicates over these clocks, and are henceforth called *clock constraints*.

Definition 5.1.1 (*Clocks*).
A clock is a variable that can assume values in the set of non-negative real numbers, represented by the set $\mathbb{R}_{\geq \mathbf{0}}$. Henceforth, we will assume an infinite set of clocks $\mathcal{C}$. Typical clock variables are c, c_1, c_2, etc.

Definition 5.1.2 (*Clock Valuation*).
A valuation, or interpretation of clocks assigns a non-negative real value to the clocks, i.e. a valuation is a mapping $\vartheta{:}\mathcal{C} \to \mathbb{R}_{\geq \mathbf{0}}$, such that for every clock $c \in \mathcal{C}$, we have $\vartheta(c) \in \mathbb{R}_{\geq \mathbf{0}}$. The set of all valuations is denoted $\mathbb{V}$, and the (singleton) set $\mathbb{V}^{\mathbf{0}}$ denotes the set with the *null-valuation* $\vartheta(c) = \mathbf{0}$ for all $c \in \mathcal{C}$ as its sole element. Since we often are interested in partial valuations, we introduce the notations $\mathbb{V}_C$ and $\mathbb{V}^{\mathbf{0}}_C$ for a set $C \subseteq \mathcal{C}$, to denote resp. the set of all valuations $\vartheta{:}C \to \mathbb{R}_{\geq \mathbf{0}}$, and and the set with the null-valuation $\vartheta(c) = \mathbf{0}$ for all $c \in C$.

Definition 5.1.3 (*Clock Constraint*).
A clock constraint is a conjunction of atomic constraints; each of these atomic constraints compares clock values with time constants. Clock constraints are thus timing constraints on one or more clocks. Formally, the set $\Phi(\mathcal{C})^*$ consists of *clock constraints* φ, which are defined by the grammar, given by Eqn. (5.1).

$$\varphi ::= c \sim v \mid (c_1 - c_2) \sim v \mid (c_1 + c_2) \sim v \mid \varphi_1 \wedge \varphi_2 \tag{5.1}$$

Here, c_1, c_2 are clocks in the set $\mathcal{C}$, v is a constant in $\mathbb{R}_{\geq \mathbf{0}}$ and $\sim \in \{<, \leq, >, \geq\}$.

The interpretation $[\![\varphi]\!]^{\vartheta}$ of a clock constraint φ in a valuation ϑ is defined inductively by (5.2). Here, we define $[\![v]\!]^{\vartheta} = v$ and $[\![c]\!]^{\vartheta} = \vartheta(c)$.

$$\begin{aligned}
[\![c \sim v]\!]^{\vartheta} &= [\![c]\!]^{\vartheta} \sim [\![v]\!]^{\vartheta} \\
[\![(c_1 - c_2) \sim v]\!]^{\vartheta} &= ([\![c_1]\!]^{\vartheta} - [\![c_2]\!]^{\vartheta}) \sim [\![v]\!]^{\vartheta} \\
[\![(c_1 + c_2) \sim v]\!]^{\vartheta} &= ([\![c_1]\!]^{\vartheta} + [\![c_2]\!]^{\vartheta}) \sim [\![v]\!]^{\vartheta} \\
[\![\varphi_1 \wedge \varphi_2]\!]^{\vartheta} &= [\![\varphi_1]\!]^{\vartheta} \wedge [\![\varphi_2]\!]^{\vartheta}
\end{aligned} \tag{5.2}$$

Satisfaction of a clock constraint φ in a valuation ϑ, denoted by $\vartheta \models \varphi$ means that the closed formula $[\![\varphi]\!]^{\vartheta}$ is evaluated to true. The set of *satisfiable* clock constraints $\Phi(\mathcal{C})$ is the set of clock constraints $\varphi \in \Phi(\mathcal{C})^*$ for which there exists a valuation ϑ, such that $\vartheta \models \varphi$. In short, $\Phi(\mathcal{C})$ is the subset of clock constraints not containing conjunctives with atomic constraints of the form $x < \mathbf{0}$. In the remainder of this thesis, we restrict ourselves to using satisfiable clock constraints.

Notation 5.1.4. We allow ourselves to write $\vartheta, \chi \models \varphi$, for arbitrary clock valuations ϑ, χ and clock constraint φ, when we mean both $\vartheta \models \varphi$ and $\chi \models \varphi$. The notation $\vartheta[\lambda := \mathbf{0}]$ for a set $\lambda \subseteq \mathcal{C}$ is a short-hand for the valuation ϑ', defined as $\vartheta'(c) = \mathbf{0}$ for all $c \in \lambda$ and $\vartheta'(c) = \vartheta(c)$ for all $c \in \mathcal{C} \setminus \lambda$.

The constants to which clocks are compared are often restricted to constants from the set of non-negative rationals. This restriction is necessary for obtaining the decidability results that allow for the automated verification of timed automata such as is offered by tools (e.g. UPPAAL [79]). When decidability is not the issue, restrictions are not necessary. Since we are not primarily interested in decidability in this chapter, we henceforth assume clocks can be compared with any element in the set of non-negative real numbers.

Notation 5.1.5. We often shall consider constants as functions; hence, the addition in e.g. $\vartheta + d$ is the addition as defined for functions, i.e. for all clocks $c \in \mathcal{C}$ and delays $d \in \mathbb{R}_{\geq \mathbf{0}}$, we have $(\vartheta + d)(c) = \vartheta(c) + d$.

A property of clock constraints that is used on various occasions is the *interpolation* property.

Property 5.1.6 (*Interpolation*).
For all valuations ϑ for clocks in $\mathcal{C}$, all clock constraints φ in $\Phi(\mathcal{C})$ and all delays $d \in \mathbb{R}_{\geq \mathbf{0}}$, implication (5.3) holds.

$$\vartheta, (\vartheta + d) \models \varphi \Rightarrow \forall_{d' \in [\mathbf{0}, d]}\ (\vartheta + d') \models \varphi \tag{5.3}$$

Proof.
Let $\vartheta \in \mathbb{V}$, and $\varphi \in \Phi(\mathcal{C})$. Assume $d \in \mathbb{R}_{\geq \mathbf{0}}$, and suppose $\vartheta, (\vartheta + d) \models \varphi$. Let $d' \in [\mathbf{0}, d]$. By structural induction, we show that also $(\vartheta + d') \models \varphi$.

Let $\sim$ be an arbitrary relation in $\{<, \leq, \geq, >\}$.

1. For clock constraints of the form $c \sim v$, we have both $\vartheta(c) \sim v$ and $\vartheta(c) \sim v - d$. Now, general rules of calculus automatically yield $\vartheta(c) \sim v - d'$ for all relations $\sim$.

2. Suppose the clock constraint φ is of the form $c_1 - c_2 \sim v$. Then, it follows that $\vartheta(c_1) - \vartheta(c_2) = (\vartheta + d')(c_1) - (\vartheta + d')(c_2)$ for arbitrary d'.

3. Thirdly, suppose the clock constraint φ is of the form $c_1 + c_2 \sim v$. We already know $\vartheta(c_1) + \vartheta(c_2) \sim v$ and $\vartheta(c_1) + \vartheta(c_2) \sim v - 2d$. Again, general rules of calculus yield $\vartheta(c_1) + \vartheta(c_2) \sim v - 2d'$ for all relations $\sim$.

4. For clock constraints of the form $\varphi_1 \wedge \varphi_2$, it clearly holds if we assume it holds for φ_1 and φ_2, given the definition of $(\vartheta + d') \models (\varphi_1 \wedge \varphi_2)$.

Thus, we know that $(\vartheta + d') \models \varphi$ holds for arbitrary clock constraints φ. □

Timed Automata are usually associated with a graphical syntax, consisting of nodes, edges and textual decorations on both nodes and edges. This graphical syntax is, for small systems, often favourable over a textual description. In proofs and larger examples, however, the graphical denotation of a timed automaton is rather clumsy and the textual notation is used. Typically, a timed automaton is defined by several sets and relations (see Def. 5.1.7).

Definition 5.1.7 (*Timed Automaton*).
Formally, a timed automaton X is a tuple $\langle L, L^0, \Sigma, C, \iota, E\rangle$,where

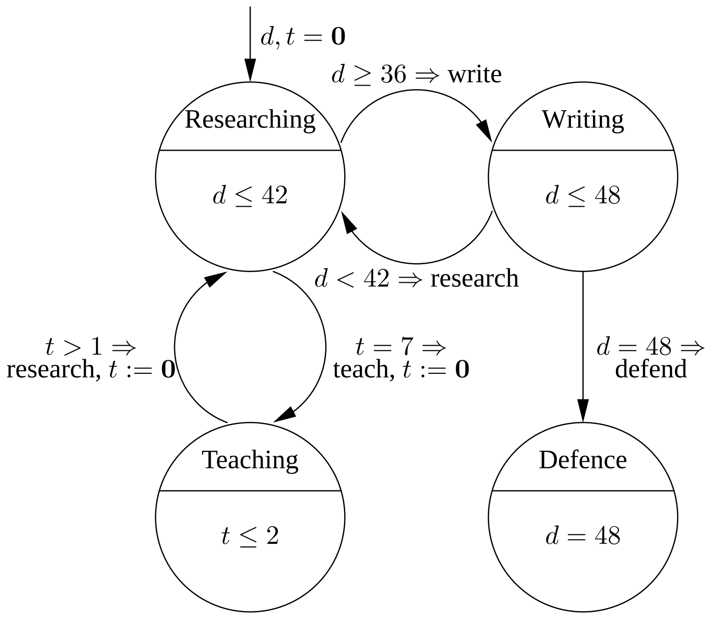
- L is a finite set of *locations*,
- $\emptyset \subset L^0 \subseteq L$ is a set of *initial* locations,
- Σ is a finite set of labels,
- $C \subseteq \mathcal{C}$ is a finite set of clocks,
- $\iota{:}L \rightarrow \Phi(C)$ is a mapping, called *invariant*, that labels each location l in L with some clock constraint in $\Phi(C)$; we require that for all initial locations $l \in L^0$, $\vartheta_{\mathbf{0}} \models \iota(l)$ for $\vartheta_{\mathbf{0}} \in \mathbb{V}_C^0$,
- $E \subseteq L \times \Sigma \times \Phi(C) \times 2^C \times L$ is a set of *switches*.

We write $l \xrightarrow{\sigma,\varphi,\lambda} l'$ for $(l, \sigma, \varphi, \lambda, l') \in E$.

Remark 5.1.8. *Restricting the invariants for initial locations to invariants that are satisfied by the null-valuation is not very common but ensures we disallow deadlocked timed automata. Since for all practical purposes, these types of automata are never interesting, we exclude them from our set of valid automata. For an investigation of timed automata with the possibility of arriving in a deadlocked state, refer to [125].*

In its graphical notation, a timed automaton is a directed graph, where nodes represent the locations and edges represent the switches. Edges are decorated with the information that is available from the set of switches; nodes contain the information pertaining to the invariant of the associated location. Before we arrive at a formal exposition of the semantics of a Timed Automaton, we briefly explain the intended meaning of a timed automaton, and provide an instructive example.

At all times, we have a notion of "being" in one of the nodes; the invariant, associated to the node describes the circumstances under which we can actually be in a given node. We can leave a node by travelling along a suitable edge. Determining whether an edge is suitable, is decided by the information found along-side an edge. This information represents an *action* σ, a *guard* φ and the *reset-set* λ. Whenever the guard φ, consisting of a clock constraint, is satisfied, the edge is "suitable" for leaving the node, and the action σ is enabled. Upon execution of σ, the clocks in the set λ are simultaneously reset to zero.



$d, t = \mathbf{0}$
$d \geq 36 \Rightarrow$ write
Researching
$d \leq 42$
Writing
$d \leq 48$
$d < 42 \Rightarrow$ research
$t > 1 \Rightarrow$ research, $t := \mathbf{0}$
$t = 7 \Rightarrow$ teach, $t := \mathbf{0}$
$d = 48 \Rightarrow$ defend
Teaching
$t \leq 2$
Defence
$d = 48$

Figure 5.1: Life-Cycle of a Ph.D. Student

Example 5.1.9. We consider the life-cycle of an average Ph.D. student. The life-span of a Ph.D. student is fixed at 48 months. Most of his/her time is spent on research activities, which can involve anything from creating art to publishing papers to organising conferences. Periodically, courses have to be taught, taking up at least a month of precious research time. The defence is, optimistically, planned in the very last month of the Ph.D. contract. A timed automaton, modelling the manufacturing of Ph.D.'s is depicted in Fig. 5.1. The automaton can serve as input to e.g. UPPAAL which in turn can be used to verify properties of the model such as deadlock and reachability. Desirable properties are e.g. "the state *Defence* is reachable" and "teaching activities do not lead to deadlocks". Of course, an in-depth analysis requires a thorough understanding of the semantics of a timed automaton, so we do not go into the validity (or falsity) of the above properties.

Modelling real-time systems is a difficult and often error-prone task, which depends highly on the size of the system measured in e.g. number of locations, switches and clocks. The means for giving a compositional description is likely to assist the modelling of more complex systems. Moreover, it is not uncommon for systems to be composed of smaller *sub-systems* (also *components*) that exchange information via communications or synchronisations. For the above-mentioned reasons, timed automata can be constructed compositionally, using the $\|_s$ operator, modelling a specific type of parallel composition. Timed Automata described as the composition of a number of other timed automata are so-named *synchronising timed automata*. Synchronising Timed Automata can be rewritten to timed automata using the set of rules given below.

Definition 5.1.10 (*Synchronising Timed Automata*).
Let $X_1 = \langle L_1, L_1^0, \Sigma_1, C_1, \iota_1, E_1 \rangle$ and $X_2 = \langle L_2, L_2^0, \Sigma_2, C_2, \iota_2, E_2 \rangle$ be timed automata, for which their sets of clocks are disjoint, i.e. $C_1 \cap C_2 = \emptyset$. The parallel composition of X_1 and X_2, denoted by $X_1 \|_s X_2$ is the timed automaton $\langle L_1 \times L_2, L_1^0 \times L_2^0, \Sigma_1 \cup \Sigma_2, C_1 \cup C_2, \iota, E \rangle$, where $\iota(l, s) = \iota_1(l) \wedge \iota_2(s)$ and the set of switches E is defined as the least relation satisfying the rules of Table 5.1.

$$\frac{l \xrightarrow{\sigma,\varphi,\lambda} l' \quad s \xrightarrow{\sigma,\psi,\kappa} s'}{(l,s) \xrightarrow{\sigma,\varphi\wedge\psi,\lambda\cup\kappa} (l',s')} \qquad \frac{l \xrightarrow{\sigma,\varphi,\lambda} l' \quad \sigma \in \Sigma_1 \setminus \Sigma_2}{(l,s) \xrightarrow{\sigma,\varphi,\lambda} (l',s)} \qquad \frac{s \xrightarrow{\sigma,\psi,\kappa} s' \quad \sigma \in \Sigma_2 \setminus \Sigma_1}{(l,s) \xrightarrow{\sigma,\psi,\kappa} (l,s')}$$

Table 5.1: Deduction Rules for Synchronisation

The operator $\|_s$ models *blocking synchronisation on shared labels*, i.e. for all labels σ shared by other component(s), a component can execute action σ iff the other component(s) can also do so. The operator $\|_s$ is, in terms of the operators of μCRL$_t$, a derived operator. To provide an intuition behind the use of synchronising timed automata, we consider the following example.

Example 5.1.11. We explore the activities of a Ph.D. student a bit further. A mandatory activity for every Ph.D. student is to participate in a number of courses. Grouped together, these courses take approx. one month to complete. Teaching, writing and research activities are postponed until the end of the course. The synchronising timed automata of Fig. 5.1 and 5.2 yield a new

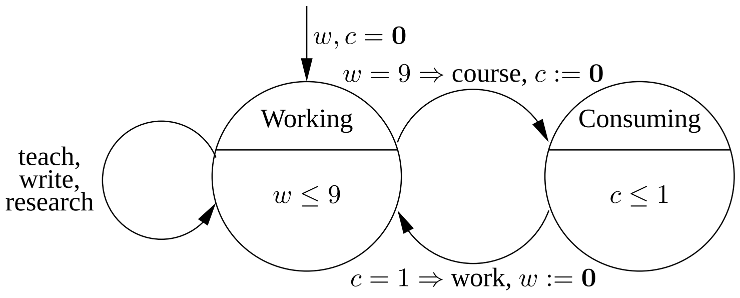
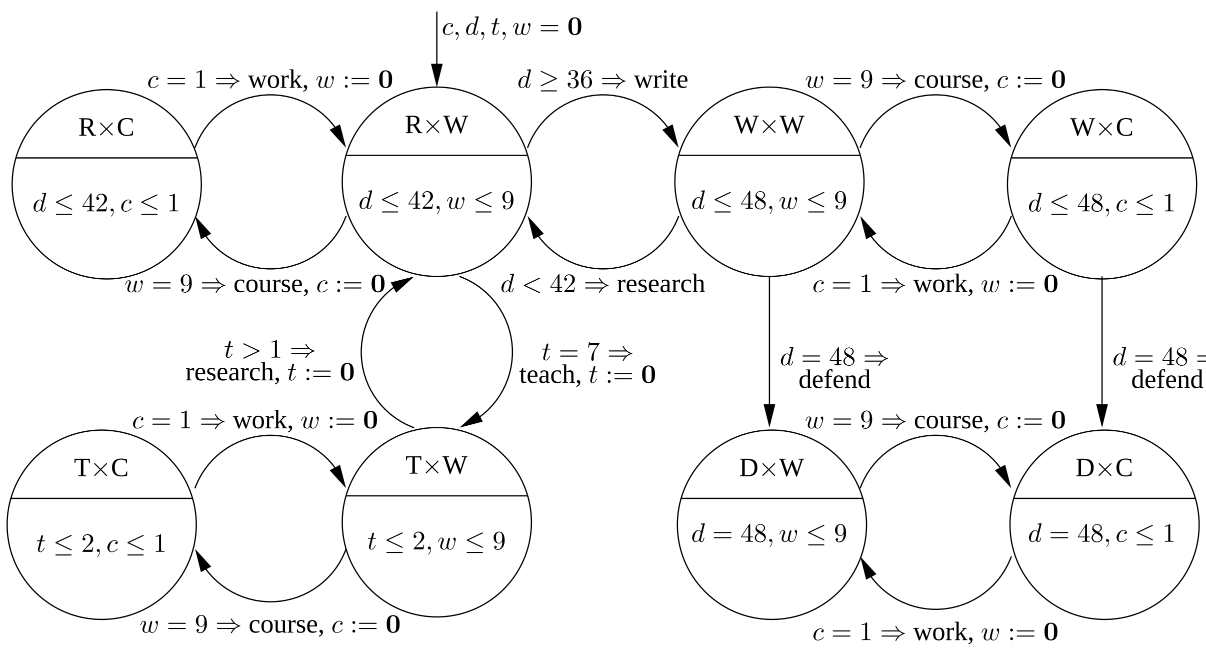

$w, c = \mathbf{0}$
$w = 9 \Rightarrow$ course, $c := \mathbf{0}$
Working
$w \leq 9$
Consuming
$c \leq 1$
teach, write, research
$c = 1 \Rightarrow$ work, $w := \mathbf{0}$

Figure 5.2: Mandatory Research Activity

automaton, depicted in Fig. 5.3. This automaton is the result of applications of the rules in Def. 5.1.10. We observe that the complexity of the composite automaton already obscures much of the information that it contains. For instance, not all locations of the composite automaton are reachable from the initial location, but it requires substantial effort to find out which.

5.1.2 Semantics — Standard

A common practice in defining the semantics of a formalism is by defining a translation of the formalism into another well-understood formalism, also referred to as *model*. Generally, *Timed Labelled Transition Systems* (TLTSs) are used as the underlying model for timed formalisms. Perhaps the most widespread TLTS is the (in this thesis referred to as the) *Two-Phase Labelled Transition System* (TPLTS), separating timed transitions (labelled with elements of the alphabet $\mathbb{R}_{\geq \mathbf{0}}$) and action transitions; the less popular TLTS (here referred to as the) *Time-Stamped Labelled Transition System* (TSLTS), was already introduced in chapter 2. Formalisms that are based on TPLTSs include timed automata [8], hybrid automata [62], various extensions of ACP with timing (e.g. [17]) and ATP [99]. As mentioned earlier, the TSLTS model is less popular, and seems to be used mainly in ACP related areas.



$c, d, t, w = \mathbf{0}$
$c = 1 \Rightarrow$ work, $w := \mathbf{0}$
$d \geq 36 \Rightarrow$ write
$w = 9 \Rightarrow$ course, $c := \mathbf{0}$
R×C
$d \leq 42, c \leq 1$
R×W
$d \leq 42, w \leq 9$
W×W
$d \leq 48, w \leq 9$
W×C
$d \leq 48, c \leq 1$
$w = 9 \Rightarrow$ course, $c := \mathbf{0}$
$d < 42 \Rightarrow$ research
$c = 1 \Rightarrow$ work, $w := \mathbf{0}$
$t > 1 \Rightarrow$ research, $t := \mathbf{0}$
$t = 7 \Rightarrow$ teach, $t := \mathbf{0}$
$d = 48 \Rightarrow$ defend
$d = 48 \Rightarrow$ defend
$c = 1 \Rightarrow$ work, $w := \mathbf{0}$
$w = 9 \Rightarrow$ course, $c := \mathbf{0}$
T×C
$t \leq 2, c \leq 1$
T×W
$t \leq 2, w \leq 9$
D×W
$d = 48, w \leq 9$
D×C
$d = 48, c \leq 1$
$w = 9 \Rightarrow$ course, $c := \mathbf{0}$
$c = 1 \Rightarrow$ work, $w := \mathbf{0}$

Figure 5.3: Fitting mandatory research activities into the Ph.D. Life-Cycle

Remark 5.1.12. *The correspondence between TPLTSs and TSLTSs is suggested in e.g. [73, 38]. An informal explanation is as follows. Under the condition of time-additivity (i.e. waiting for d time units and then immediately waiting for another d' time units cannot be distinguished from waiting $d + d'$ time units at once), we have the following intuition. A time-stamped transition $s \xrightarrow{\sigma}_t s'$ in a TSLTS corresponds to a combination of an appropriate delay transition $s \overset{t}{\longmapsto} s''$ to a certain state s'', followed by an action transition $s'' \xrightarrow{\sigma} s'$ in a TPLTS. The delay predicate $\mathcal{U}_t(s)$ in a TSLTS, meaning that a process can idle until absolute point of time t while in state s, corresponds to the existence of a state s' in a TPLTS, for which we have $s \overset{t}{\longmapsto} s'$.*

In this section, we define the semantics of a timed automaton using the underlying model of TPLTSs. We deviate slightly from the standard definition of the semantics for a timed automaton by working in absolute time rather than relative time. The embedding of relative time into absolute time, however, is straightforward. On the level of a timed automaton, we can think of “now” in absolute time as the value of a redundant clock t, i.e. a clock that is never reset nor referred to in any of the automaton’s clock constraints.

Definition 5.1.13 (*Semantics of a Timed Automaton (TPLTS)*).
Let $X = \langle L, L^0, \Sigma, C, \iota, E\rangle$ be a timed automaton. The TPLTS X^s, associated to the timed automaton X is $X^s = \langle L \times \mathbb{V} \times \mathbb{R}_{\geq \mathbf{0}}, L^0 \times \mathbb{V}_C^{\mathbf{0}} \times \{\mathbf{0}\}, \Sigma, \rightarrow, \mapsto\rangle$. The action transitions $\rightarrow$ and the timed transitions $\mapsto$ are defined as the least relations satisfying the rules of Table 5.2.
The state-space of X^s consists of the set of states that can be reached from the initial states.

Remark 5.1.14. *The interpretation given here allows for multiple events in zero time (so-named urgent actions). Not all interpretations allow for these urgent actions. Most notably, the original definition of Alur and Dill [8] requires the passage of (non-zero) time between two successive action transitions.*

$$\frac{\vartheta, (\vartheta + d) \models \iota(l) \quad \mathbf{0} \leq d}{(l, \vartheta, t) \overset{t+d}{\longmapsto} (l, \vartheta + d, t + d)}$$

$$\frac{l \xrightarrow{\sigma,\varphi,\lambda} l' \quad \vartheta \models \varphi \quad \vartheta \models \iota(l) \quad \vartheta' = \vartheta[\lambda := 0] \quad \vartheta' \models \iota(l')}{(l, \vartheta, t) \xrightarrow{\sigma} (l', \vartheta', t)}$$

Table 5.2: Deduction Rules for Passage of Time and Single-step Evolution

We distinguish timed automata on the basis of their behaviour, using a timed bisimulation. We use bisimulation, as it is the strongest equivalence that differentiates the behaviour of two systems on the basis of their behaviour, rather than on their syntax. Moreover, any results, obtained under the assumption of using timed bisimulation immediately apply to weaker notions of equivalence. For an overview of other equivalences, see [48] for a discussion on equivalence for general specification formalisms, and [38] for equivalences tailored to the framework of timed automata.

Definition 5.1.15 (*Timed Bisimulation for Timed Automata (TPLTS)*).
Let X_1 and X_2 be two timed automata, and let X_1^s and X_2^s be the interpretations of resp. X_1 and X_2 in terms of TPLTSs.
A relation $\mathcal{R} \subseteq (L_1 \times \mathbb{V}_{C_1} \times \mathbb{R}_{\geq \mathbf{0}}) \times (L_2 \times \mathbb{V}_{C_2} \times \mathbb{R}_{\geq \mathbf{0}})$ is a timed bisimulation iff for all states (l_1, ϑ, t) and (l_2, χ, u), the following properties are satisfied.

1. if $(l_1, \vartheta, t)\mathcal{R}(l_2, \chi, u)$ and $(l_1, \vartheta, t) \xrightarrow{\sigma} (l_1', \vartheta', t)$ for some $\sigma \in \Sigma_1$, then there exists a state (l_2', χ', u), such that $(l_2, \chi, u) \xrightarrow{\sigma} (l_2', \chi', u)$ and $(l_1', \vartheta', t)\mathcal{R}(l_2', \chi', u)$. Similarly for each action transition of automaton X_2.
2. if $(l_1, \vartheta, t)\mathcal{R}(l_2, \chi, u)$ and $(l_1, \vartheta, t) \overset{t+d}{\longmapsto} (l_1, \vartheta', t + d)$ for some $d \in \mathbb{R}_{\geq \mathbf{0}}$, then there exists a state $(l_2, \chi', u + d)$, such that $(l_2, \chi, u) \overset{u+d}{\longmapsto} (l_2, \chi', u + d)$ and $(l_1, \vartheta', t + d)\mathcal{R}(l_2, \chi', u + d)$. Similarly for each delay transition of automaton X_2.
3. For each state $(l_1, \vartheta_\mathbf{0}, \mathbf{0})$ with $l_1 \in L_1^0$ and $\vartheta_\mathbf{0} \in \mathbb{V}_{C_1}^\mathbf{0}$, there exists a state $(l_2, \chi_\mathbf{0}, \mathbf{0})$, such that $l_2 \in L_2^0$, $\chi_\mathbf{0} \in \mathbb{V}_{C_2}^\mathbf{0}$ and $(l_1, \vartheta_\mathbf{0}, \mathbf{0})\mathcal{R}(l_2, \chi_\mathbf{0}, \mathbf{0})$, and similarly for each initial state of X_2.

If there exists a relation $\mathcal{R}$, relating all states between X_1^s and X_2^s according to the above-listed criteria, then we say the two timed automata X_1 and X_2 are timed bisimilar, denoted by $\mathcal{R}{:}X_1 \leftrightarroweq{}_t^{\iota} X_2$.

As a result of this definition, two timed bisimilar timed automata agree on their elapsed time for all reachable and timed bisimilar states (see Lemma 5.1.16).

Lemma 5.1.16. Let X_1 and X_2 be two timed bisimilar timed automata and $\mathcal{R}$ be a relation relating X_1 and X_2 , i.e. $\mathcal{R}{:}X_1 \leftrightarroweq{}_t^{\iota} X_2$. Then, for all states (l, ϑ, t) of X_1^s and (s, χ, u) of X_2^s, reachable from the initial states of X_1 and X_2, implication (5.4) holds.

$$(l, \vartheta, t)\mathcal{R}(s, \chi, u) \Rightarrow u = t \tag{5.4}$$

Proof. Let $\mathcal{R}$ be a timed bisimulation relation, relating the timed automata X_1 and X_2. We proceed by induction on the number of transitions taken from an initial state.
Clearly, implication (5.4) holds for initial states of the systems. Suppose we have the reachable states (l, ϑ, t) and (s, χ, u), such that $(l, \vartheta, t)\mathcal{R}(s, \chi, u)$ and $u = t$. Now, we can distinguish four cases, two of which we investigate below. The remaining two cases are symmetric to the below discussed cases and are therefore omitted.

1. Suppose $(l, \vartheta, t) \xrightarrow{\sigma} (l', \vartheta', t)$. Since $\mathcal{R}$ is a timed bisimulation relation, relating X_1 and X_2, we also have $(s, \chi, u) \xrightarrow{\sigma} (s', \chi', u)$ and $(l', \vartheta', t)\mathcal{R}(s', \chi', u)$ for some state (s', χ', u). We still have $u = t$, and therefore, implication (5.4) holds.

2. Suppose $(l, \vartheta, t) \overset{t+d}{\longmapsto} (l, \vartheta', t+d)$. Again, since $\mathcal{R}$ is a timed bisimulation relation, we also have $(s, \chi, u) \overset{u+d}{\longmapsto} (s, \chi', u+d)$ and $(l, \vartheta', t+d)\mathcal{R}(s, \chi', u+d)$ for some state $(s, \chi', u+d)$. Since we have $u = t$, we also know $u + d = t + d$, confirming implication (5.4).

$\square$

Lemma 5.1.17. Let $X = \langle L, L^0, \Sigma, C, \iota, E\rangle$ be a timed automaton. Then, for all states (l, ϑ, t) of X^s, reachable from an initial state of X, $\vartheta \models \iota(l)$ holds.

Proof. We again provide a proof by means of induction on the number of transitions taken from an initial state.
By definition, Lemma 5.1.17 holds for initial states, since initial states have invariants satisfied by the null-valuation. Suppose we have a reachable state (l, ϑ, t), such that $\vartheta \models \iota(l)$. Then, there are two options.

1. Let $(l, \vartheta, t) \overset{t+d}{\longmapsto} (l, \vartheta + d, t + d)$ for some $d \geq \mathbf{0}$. Then, by definition, we must have $\vartheta, \vartheta + d \models \iota(l)$ and therefore surely also $\vartheta + d \models \iota(l)$.

2. Let $(l, \vartheta, t) \xrightarrow{\sigma} (l', \vartheta', t)$. By definition, this is possible, only when $\vartheta' \models \iota(l')$.

Both options agree with Lemma 5.1.17. $\square$

Apart from a semantics for a timed automaton, we also provide a semantics for the parallel operator $\|_s$. This shows the compositionality of the operator.

Definition 5.1.18 (*Semantics of Synchronising Timed Automata (TPLTS)*).
Let $X_1 = \langle L_1, L_1^0, \Sigma_1, C_1, \iota_1, E_1\rangle$ and $X_2 = \langle L_2, L_2^0, \Sigma_2, C_2, \iota_2, E_2\rangle$ be timed automata. The semantics of $X_1\|_s X_2$ is given by the TPLTS $(X_1)^s\|_s(X_2)^s = \langle (L_1 \times \mathbb{V}_{C_1} \times \mathbb{R}_{\geq \mathbf{0}}) \times (L_2 \times \mathbb{V}_{C_2} \times \mathbb{R}_{\geq \mathbf{0}}), (L_1^0 \times \mathbb{V}^{\mathbf{0}}_{C_1} \times \{\mathbf{0}\}) \times (L_2^0 \times \mathbb{V}^{\mathbf{0}}_{C_2} \times \{\mathbf{0}\}), \Sigma_1 \cup \Sigma_2, \rightarrow, \mapsto\rangle$, where for all states $(l, \vartheta, t) \in X_1^s$ and $(s, \chi, u) \in X_2^s$, the set of action and delay transitions $\rightarrow$ and $\mapsto$ are defined as the least relations satisfying the rules of Table 5.3.

The rules for constructing a new timed automaton, based on the parallel composition of two timed automata, described in Def. 5.1.10 are in accordance with the semantical interpretation presented in Def. 5.1.18 (see the next Lemma).

$$\frac{(l,\vartheta,t) \overset{t+d}{\longmapsto} (l,\vartheta+d,t+d) \quad (s,\chi,t) \overset{t+d}{\longmapsto} (s,\chi+d,t+d)}{(l,\vartheta,t)\|_s(s,\chi,t) \overset{t+d}{\longmapsto} (l,\vartheta+d,t+d)\|_s(s,\chi+d,t+d)}$$

$$\frac{(l,\vartheta,t) \xrightarrow{\sigma} (l',\vartheta',t) \quad (s,\chi,t) \overset{t}{\longmapsto} (s,\chi,t) \quad \sigma \in \Sigma_1 \setminus \Sigma_2}{(l,\vartheta,t)\|_s(s,\chi,t) \xrightarrow{\sigma} (l',\vartheta',t)\|_s(s,\chi,t)}$$

$$\frac{(s,\chi,u) \xrightarrow{\sigma} (s',\chi',u) \quad (l,\vartheta,u) \overset{u}{\longmapsto} (l,\vartheta,u) \quad \sigma \in \Sigma_2 \setminus \Sigma_1}{(l,\vartheta,u)\|_s(s,\chi,u) \xrightarrow{\sigma} (l,\vartheta,u)\|_s(s',\chi',u)}$$

$$\frac{(l,\vartheta,t) \xrightarrow{\sigma} (l',\vartheta',t) \quad (s,\chi,t) \xrightarrow{\sigma} (s',\chi',t) \quad \sigma \in \Sigma_1 \cap \Sigma_2}{(l,\vartheta,t)\|_s(s,\chi,t) \xrightarrow{\sigma} (l',\vartheta',t)\|_s(s',\chi',t)}$$

Table 5.3: Rules for the Synchronisation Operator

$$\begin{tikzcd}
X_1 \parallel_s X_2 \arrow[r, dash] \arrow[d, dash] & (X_1 \parallel_s X_2)^s \\
(X_1)^s \parallel_s (X_2)^s \arrow[ur, shift left, leftrightarrow, "\iota"] \arrow[ur, shift right, dash, "\not\iota"'] &
\end{tikzcd}$$

Figure 5.4: Soundness of Synchronising Timed Automata

Lemma 5.1.19. Let X_1 and X_2 be two timed automata, and let X_1^s and X_2^s be the TPLTS associated to X_1 and X_2, then there exists a timed bisimulation relation $\mathcal{R}$, such that (5.5) holds (see also Fig. 5.4).

$$\mathcal{R}{:}X_1^s \|_s X_2^s \;\underline{\leftrightarrow}_t^{\iota}\; (X_1 \|_s X_2)^s \tag{5.5}$$

Proof. Let X_1 and X_2 be two timed automata. We define a relation $\mathcal{R}$ and make it plausible that this relation is a timed bisimulation relation relating $X_1^s \|_s X_2^s$ and $(X_1 \|_s X_2)^s$.

$$\mathcal{R} = \{\langle\, (l, \vartheta, t) \|_s (s, \chi, t), ((l, s), \vartheta \cup \chi, t) \,\rangle \mid l \in L_1, s \in L_2, \vartheta \in \mathbb{V}_{C_1}, \vartheta \in \mathbb{V}_{C_2}, t \in \mathbb{R}_{\geq \mathbf{0}}\} \tag{5.6}$$

For $\mathcal{R}$ to be a timed bisimulation relation, the requirements of Def. 5.1.15 have to be checked. The requirement concerning initial states is satisfied, which is not hard to see. We therefore focus on the delay transitions and the action transitions.

1. Assume $(l, \vartheta, t) \|_s (s, \chi, t) \mathcal{R} ((l, s), \vartheta \cup \chi, t)$, and suppose $(l, \vartheta, t) \|_s (s, \chi, t) \xmapsto{t+d} (l, \vartheta + d, t + d) \|_s (s, \chi + d, t + d)$. Then, also $(l, \vartheta, t) \xmapsto{t+d} (l, \vartheta + d, t + d)$ and $(s, \chi, t) \xmapsto{t+d} (s, \chi + d, t + d)$. Hence, also $\vartheta, (\vartheta + d) \models \iota_1(l)$ and $\chi, (\chi + d) \models \iota_2(s)$. Since clocks in X_1 and X_2 are disjoint, we can derive $\vartheta \cup \chi, ((\vartheta \cup \chi) + d) \models \iota(l, s)$, which is sufficient for obtaining $((l, s), \vartheta \cup \chi, t) \xmapsto{t+d} ((l, s), (\vartheta \cup \chi) + d, t + d)$. By construction of the relation $\mathcal{R}$, we have $(l, \vartheta + d, t + d) \|_s (s, \chi + d, t + d) \mathcal{R} ((l, s), (\vartheta \cup \chi) + d, t + d)$. By following the above steps in reverse it follows that from $((l, s), \vartheta \cup \chi, t) \xmapsto{t+d} ((l, s), (\vartheta \cup \chi) + d, t + d)$ we can derive $(l, \vartheta, t) \|_s (s, \chi, t) \xmapsto{t+d} (l, \vartheta + d, t + d) \|_s (s, \chi + d, t + d)$.

2. Suppose $(l, \vartheta, t) \|_s (s, \chi, t) \mathcal{R} ((l, s), \vartheta \cup \chi, t)$; assume $\sigma \in \Sigma_1 \cap \Sigma_2$ and suppose we have $(l, \vartheta, t) \|_s (s, \chi, t) \xrightarrow{\sigma} (l', \vartheta', t) \|_s (s', \chi', t)$. Therefore, also $(l, \vartheta, t) \xrightarrow{\sigma} (l', \vartheta', t)$ and $(s, \chi, t) \xrightarrow{\sigma} (s', \chi', t)$. This means we have $\vartheta \models \iota_1(l)$, $\chi \models \iota_2(s)$, $\vartheta' \models \iota_1(l')$ and $\chi' \models \iota_2(s')$, and, moreover, there are switches $l \xrightarrow{\sigma, \varphi, \lambda} l'$ and $s \xrightarrow{\sigma, \psi, \kappa} s'$, such that $\vartheta \models \varphi$, $\chi \models \psi$, $\vartheta' = \vartheta[\lambda := \mathbf{0}]$ and $\chi' = \chi[\kappa := \mathbf{0}]$. Given the assumption $C_1 \cap C_2 = \emptyset$, we can then identify a switch $(l, s) \xrightarrow{\sigma, \varphi \wedge \psi, \lambda \cup \kappa} (l', s')$, such that $\vartheta \cup \chi \models \iota(l, s)$, $\vartheta' \cup \chi' \models \iota(l', s')$ and, finally, $\vartheta' \cup \chi' = (\vartheta \cup \chi)[\lambda \cup \kappa := \mathbf{0}]$. Therefore, also $((l, s), \vartheta \cup \chi, t) \xrightarrow{\sigma} ((l', s'), \vartheta' \cup \chi', t)$ and by construction, we have $(l', \vartheta', t) \|_s (s', \chi', t) \mathcal{R} ((l', s'), \vartheta' \cup \chi', t)$. The above steps in reverse lead to the symmetric case where $((l, s), \vartheta \cup \chi, t) \xrightarrow{\sigma} ((l', s'), \vartheta' \cup \chi', t)$ leads to $(l, \vartheta, t) \|_s (s, \chi, t) \mathcal{R} ((l, s), \vartheta \cup \chi, t)$.

3. Suppose $(l, \vartheta, t) \|_s (s, \chi, t) \mathcal{R} ((l, s), \vartheta \cup \chi, t)$; assume $\sigma \in \Sigma_1 \setminus \Sigma_2$ and suppose we have $(l, \vartheta, t) \|_s (s, \chi, t) \xrightarrow{\sigma} (l', \vartheta', t) \|_s (s, \chi, t)$. Therefore, we also have $(l, \vartheta, t) \xrightarrow{\sigma} (l', \vartheta', t)$ and $(s, \chi, t) \xmapsto{t} (s, \chi, t)$. We therefore have $\vartheta \models \iota_1(l)$, $\vartheta' \models \iota_1(l')$ and $\chi \models \iota_2(s)$, and, moreover, there is a switch $l \xrightarrow{\sigma, \varphi, \lambda} l'$ such that $\vartheta \models \varphi$ and $\vartheta' = \vartheta[\lambda := \mathbf{0}]$. Combining this, we can identify a switch $(l, s) \xrightarrow{\sigma, \varphi, \lambda} (l', s)$, such that $\vartheta \cup \chi \models \iota(l, s)$, $\vartheta' \cup \chi \models \iota(l', s)$, $\vartheta \cup \chi \models \varphi$ and $\vartheta' \cup \chi = (\vartheta \cup \chi)[\lambda := \mathbf{0}]$, leading to $((l, s), \vartheta \cup \chi, t) \xrightarrow{\sigma} ((l', s), \vartheta' \cup \chi, t)$ and by construction $(l', \vartheta', t) \|_s (s, \chi, t) \mathcal{R} ((l', s), \vartheta' \cup \chi, t)$. For the symmetric case and the case where $\sigma \in \Sigma_2 \setminus \Sigma_1$, similar arguments hold.

$\square$

5.1.3 Semantics — Alternative

The previous section dealt with the semantics of timed automata based on the model of TPLTSs. Focusing on the framework of μCRL$_t$, an alternative semantics, based on *Time-Stamped Labelled Transition Systems* (TSLTSs) is preferable. The semantics we define in this section uses the underlying model of TSLTSs. For an intuition behind the correspondence between the TSLTS model and the TPLTS model, refer to Remark 5.1.12.

Definition 5.1.20 (*Semantics of a Timed Automaton (TSLTS)*).
Let $X = \langle L, L^0, \Sigma, C, \iota, E\rangle$ be a timed automaton. The semantics of X is given by the TSLTS $X^{\overline{s}} = \langle L \times \mathbb{V} \times \mathbb{R}_{\geq \mathbf{0}}, L^0 \times \mathbb{V}_C^0 \times \{\mathbf{0}\}, \Sigma, \rightarrow, \mathcal{U}\rangle$. The time-stamped action transition $\rightarrow$ and the delay predicate $\mathcal{U}$ are defined as the least relations satisfying the rules of Table 5.4. The

$$\frac{\mathcal{U}_u(l,\vartheta,t) \quad u' \leq u}{\mathcal{U}_{u'}(l,\vartheta,t)} \qquad \frac{\vartheta, (\vartheta + d) \models \iota(l) \quad \mathbf{0} \leq d}{\mathcal{U}_{t+d}(l,\vartheta,t)}$$

$$\frac{\begin{array}{c} l \xrightarrow{\sigma,\varphi,\lambda} l' \quad \vartheta' = (\vartheta+d)[\lambda := 0] \quad \mathbf{0} \leq d \\ \vartheta, (\vartheta+d) \models \iota(l) \quad (\vartheta + d) \models \varphi \quad \vartheta' \models \iota(l') \end{array}}{(l,\vartheta,t) \xrightarrow{\sigma}_{t+d} (l', \vartheta', t+d)}$$

Table 5.4: Deduction Rules for Single-Step Evolution and Idling Behaviour

state-space of $X^{\overline{s}}$ consists of the set of states that can be reached from the initial states.

Remark 5.1.21. *Whenever we refer to the time-stamped semantics of a timed automaton X, we will write $X^{\overline{s}}$.*

The delay predicate $\mathcal{U}$ is defined using two distinct rules; the downward delay-closure rule is needed to ensure delaying is a time-convex activity. The presence of this rule also has an impact on the length of proofs, as we need to explicitly make a distinction between the two cases. The following Lemma states that, whenever we restrict our reasoning to *reachable* states, it suffices to prove only one case rather than both cases, as for reachable states, the downward delay-closure always holds.

Lemma 5.1.22. Let X be a timed automaton. Then, for each state (l, ϑ, t), reachable from an initial state, $\mathcal{U}_u(l, \vartheta, t)$ holds for all $u \leq t$.

Proof. Clearly, for the initial states, the statement holds; suppose state (l, ϑ, t) is reachable from an initial state and we have $\mathcal{U}_u(l, \vartheta, t)$ for all $u \leq t$. Suppose $(l, \vartheta, t) \xrightarrow{\sigma}_{u'} (l', \vartheta', u')$. Clearly, then at least the condition $\vartheta' \models \iota(l')$ must have been met. This condition, however, is sufficient for proving $\mathcal{U}_{u'}(l', \vartheta', u')$. Using the downward delay-closure rule, we then immediately have $\mathcal{U}_{t'}(l', \vartheta', u')$ for all $t' \leq u'$. □

Definition 5.1.23 (*Timed Bisimulation for Timed Automata (TSLTS)*).
Let X_1 and X_2 be two timed automata and let $X_1^{\overline{s}}$ and $X_2^{\overline{s}}$ be the interpretations of X_1 and X_2 in terms of TSLTSs.
A relation $\mathcal{R} \subseteq (L_1 \times \mathbb{V}_{C_1} \times \mathbb{R}_{\geq \mathbf{0}}) \times (L_2 \times \mathbb{V}_{C_2} \times \mathbb{R}_{\geq \mathbf{0}})$ is a timed bisimulation iff for all states (l_1, ϑ, t) and (l_2, χ, u), the following requirements are fulfilled.

1. if $(l_1, \vartheta, t)\mathcal{R}(l_2, \chi, u)$ and $(l_1, \vartheta, t) \xrightarrow{\sigma}_r (l_1', \vartheta', r)$ for some $\sigma \in \Sigma_1$ and $r \in \mathbb{R}_{\geq \mathbf{0}}$, then there exists a state (l_2', χ', u), such that $(l_2, \chi, u) \xrightarrow{\sigma}_r (l_2', \chi', r)$ and $(l_1', \vartheta', r)\mathcal{R}(l_2', \chi', r)$. Similarly for each action transition of automaton X_2.

2. if $(l_1, \vartheta, t)\mathcal{R}(l_2, \chi, u)$ and $\mathcal{U}_r(l_1, \vartheta, t)$ for some $r \in \mathbb{R}_{\geq \mathbf{0}}$, then also $\mathcal{U}_r(l_2, \chi, u)$.

3. For each state $(l_1, \vartheta_{\mathbf{0}}, \mathbf{0})$ with $l_1 \in L_1^0$ and $\vartheta_{\mathbf{0}} \in \mathbb{V}_{C_1}^{\mathbf{0}}$, there exists a state $(l_2, \chi_{\mathbf{0}}, \mathbf{0})$, such that $l_2 \in L_2^0$, $\chi_{\mathbf{0}} \in \mathbb{V}_{C_2}^{\mathbf{0}}$ and $(l_1, \vartheta_{\mathbf{0}}, \mathbf{0})\mathcal{R}(l_2, \chi_{\mathbf{0}}, \mathbf{0})$, and similarly for each initial state of X_2.

If there exists a relation $\mathcal{R}$, relating all states between $X_1^{\overline{s}}$ and $X_2^{\overline{s}}$ according to the above-listed criteria, then we say the two timed automata X_1 and X_2 are timed bisimilar, denoted by $\mathcal{R}{:}X_1 \leftrightarrow_t X_2$. When we are not particularly interested in the witnessing relation $\mathcal{R}$, we often omit it and write $X_1 \leftrightarrow_t X_2$.

As a result of this definition, two timed bisimilar timed automata synchronise on their elapsed time for all their reachable and bisimilar states (see Lemma 5.1.24).

Lemma 5.1.24. Let X_1 and X_2 be two timed bisimilar timed automata and $\mathcal{R}$ be a relation relating X_1 and X_2 , i.e. $\mathcal{R}{:}X_1 \leftrightarrow_t X_2$. Then, for all states (l, ϑ, t) of $X_1^{\overline{s}}$ and (s, χ, u) of $X_2^{\overline{s}}$, reachable from the initial states of X_1 and X_2 by means of a series of time-stamped action transitions, implication (5.7) holds.

$$(l, \vartheta, t)\mathcal{R}(s, \chi, u) \Rightarrow u = t \tag{5.7}$$

Proof. The proof proceeds according to the lines of the proof of Lemma 5.1.16. □

Definition 5.1.25 (*Semantics of Synchronising Timed Automata (TSLTS)*).
Let $X_1 = \langle L_1, L_1^0, \Sigma_1, C_1, \iota_1, E_1\rangle$ and $X_2 = \langle L_2, L_2^0, \Sigma_2, C_2, \iota_2, E_2\rangle$ be timed automata. The semantics of $X_1\|_s X_2$ is given by the TSLTS $(X_1)^{\overline{s}}\|_s(X_2)^{\overline{s}} = \langle (L_1 \times \mathbb{V}_{C_1} \times \mathbb{R}_{\geq \mathbf{0}}) \times (L_2 \times \mathbb{V}_{C_2} \times \mathbb{R}_{\geq \mathbf{0}}), (L_1^0 \times \mathbb{V}_{C_1}^{\mathbf{0}} \times \{\mathbf{0}\}) \times (L_2^0 \times \mathbb{V}_{C_2}^{\mathbf{0}} \times \{\mathbf{0}\}), \Sigma_1 \cup \Sigma_2, \rightarrow, \mathcal{U}\rangle$, where for all states $(l, \vartheta, t) \in X_1^s$ and $(s, \chi, u) \in X_2^s$, the set of time-stamped action transitions $\rightarrow$ and the delay predicate $\mathcal{U}$ are defined as the least relations satisfying the rules of Table 5.5.

Lemma 5.1.22 carries over straightforwardly to the setting of synchronising timed automata.

Lemma 5.1.26. Let X and Y be timed automata. Then, for each state $(l, \vartheta, t)\|_s(s, \chi, u)$, reachable from an initial state, $\mathcal{U}_{u'}((l, \vartheta, t)\|_s(s, \chi, u))$ holds for all $u' \leq t \max u$.

Proof. For the initial states, the above Lemma obviously holds. Suppose state $(l, \vartheta, t)\|_s(s, \chi, u)$ is reachable from an initial state and we have, for all $r \leq t \max u$, $\mathcal{U}_r((l, \vartheta, t)\|_s(s, \chi, u))$. We make a distinction between two cases.
Suppose we can reach, via a non-shared time-stamped action transition, state $(l', \vartheta', t')\|_s(s, \chi, u)$. Then, we know $u \leq t', t \leq t'$ and $\mathcal{U}_{t'}(s, \chi, u)$. Moreover, $(l, \vartheta, t) \xrightarrow{\sigma}_{t'} (l', \vartheta', t')$ is possible. This must be a reachable state, so also $\mathcal{U}_{t'}(l', \vartheta', t')$. But then also $\mathcal{U}_{t'}((l', \vartheta', t')\|_s(s, \chi, u))$. We then also have $\mathcal{U}_{u'}((l', \vartheta', t')\|_s(s, \chi, u))$ for all $u' \leq t' \max u$.
Suppose we can reach the state $(l', \vartheta', t')\|_s(s', \chi', t')$ via a shared time-stamped action transition. Then, we know $(l, \vartheta, t) \xrightarrow{\sigma}_{t'} (l', \vartheta', t')$ and $(s, \chi, t) \xrightarrow{\sigma}_{t'} (s', \chi', t')$. These states must be reachable from the initial states, so also $\mathcal{U}_{t'}(l', \vartheta', t')$ and $\mathcal{U}_{t'}(s', \chi', t')$ and therefore also $\mathcal{U}_{u'}((l', \vartheta', t')\|_s(s', \chi', t'))$ for all $u' \leq t' \max t'$. □

$$\frac{\mathcal{U}_r(l,\vartheta,t) \quad \mathcal{U}_r(s,\chi,u)}{\mathcal{U}_r((l,\vartheta,t)\|_s(s,\chi,u))}$$

$$\frac{(l,\vartheta,t) \xrightarrow{\sigma}_r (l',\vartheta',r) \quad \mathcal{U}_r(s,\chi,u) \quad \sigma \in \Sigma_1 \setminus \Sigma_2 \quad u \le r}{(l,\vartheta,t)\|_s(s,\chi,u) \xrightarrow{\sigma}_r (l',\vartheta',r)\|_s(s,\chi,u)}$$

$$\frac{(s,\chi,u) \xrightarrow{\sigma}_r (s',\chi',r) \quad \mathcal{U}_r(l,\vartheta,t) \quad \sigma \in \Sigma_2 \setminus \Sigma_1 \quad t \le r}{(l,\vartheta,t)\|_s(s,\chi,u) \xrightarrow{\sigma}_r (l,\vartheta,t)\|_s(s',\chi',r)}$$

$$\frac{(l,\vartheta,t) \xrightarrow{\sigma}_r (l',\vartheta',r) \quad (s,\chi,u) \xrightarrow{\sigma}_r (s',\chi',r) \quad \sigma \in \Sigma_1 \cap \Sigma_2}{(l,\vartheta,t)\|_s(s,\chi,u) \xrightarrow{\sigma}_r (l',\vartheta',r)\|_s(s',\chi',r)}$$

Table 5.5: Rules for the Synchronisation Operator

The rules for constructing a new timed automaton, based on the parallel composition of two timed automata, described in Def. 5.1.10 are in accordance with the semantic interpretation presented in Def. 5.1.25.

Lemma 5.1.27. Let X_1 and X_2 be two timed automata, and let $X_1^{\overline{s}}$ and $X_2^{\overline{s}}$ be the TSLTS associated to X_1 and X_2, then there exists a timed bisimulation relation $\mathcal{R}$, such that (5.8) holds (see also Fig. 5.5).

$$\mathcal{R} : X_1^{\overline{s}} \|_s X_2^{\overline{s}} \ \underline{\leftrightarrow}_t \ (X_1 \|_s X_2)^{\overline{s}} \tag{5.8}$$

$$\begin{tikzcd} X_1 \parallel_s X_2 \arrow[r, dash] \arrow[d, dash] & (X_1 \parallel_s X_2)^{\overline{s}} \\ (X_1)^{\overline{s}} \parallel_s (X_2)^{\overline{s}} \arrow[ur, leftrightarrow, "\underline{\leftrightarrow}_t"] & \end{tikzcd}$$

Figure 5.5: Soundness of Synchronising Timed Automata

Proof. The proof is essentially similar to the proof for Lemma 5.1.19. Let X_1 and X_2 be two timed automata. We define a relation $\mathcal{R}'$ and prove that its reachable subset satisfies the definition of a timed bisimulation relation.

$$\begin{array}{rcl} \mathcal{R}' & = & \{\langle (l,\vartheta,t)\|_s(s,\chi,u)\ ,\ ((l,s),\vartheta \cup (\chi + t - u),t)\rangle \ | u \le t\} \\ & \cup & \{\langle (l,\vartheta,t)\|_s(s,\chi,u)\ ,\ ((l,s),(\vartheta + u - t) \cup \chi,u)\rangle \ | t \le u\} \end{array} \tag{5.9}$$

We define $\mathcal{R} = \mathcal{R}' \cap ((R_{X_1} \times R_{X_2}) \times R_{X_1\|_s X_2})$, where R_Y represents the set of reachable states of automaton Y. The proof that $\mathcal{R}$ is indeed a timed bisimulation relation proceeds as follows. Again, for initial states, it is straightforward to verify the conditions for $\mathcal{R}$ to be a timed bisimulation relation are met. Suppose we have $(l,\vartheta,t)\|_s(s,\chi,u)\mathcal{R}((l,s),\vartheta\cup(\chi+t-u),t)$ and $u \leq t$. We distinguish three cases:

1. Assume we know $\mathcal{U}_{u'}((l,\vartheta,t)\|_s(s,\chi,u))$ for given $u' \in \mathbb{R}_{\geq \mathbf{0}}$. By definition, we then have $\mathcal{U}_{u'}(l,\vartheta,t)$ and $\mathcal{U}_{u'}(s,\chi,u)$. Based on the semantics of a timed automaton, we thus know that $\vartheta,(\vartheta+u'-t) \models \iota_1(l)$ and $\chi,(\chi+u'-u) \models \iota_2(s)$. Given that $u \leq t$, we can use the interpolation property 5.1.6 to obtain $(\chi+t-u),((\chi+t-u)+u'-t) \models \iota_2(s)$. This allows us to derive that also $\mathcal{U}_{u'}((l,s),\vartheta\cup(\chi+t-u),t)$ must hold.

   Conversely, assume we know $\mathcal{U}_{u'}((l,s),\vartheta\cup(\chi+t-u),t)$ for given $u' \in \mathbb{R}_{\geq \mathbf{0}}$. By definition, we then have $\vartheta\cup(\chi+t-u),((\vartheta\cup(\chi+t-u))+u'-t) \models \iota(l,s)$. Hence, we can also derive $\vartheta,(\vartheta+u'-t) \models \iota_1(l)$ and $\chi+t-u,\chi+u'-u \models \iota_2(s)$. Now, we use the assumption that $(l,\vartheta,t)\|_s(s,\chi,u)\mathcal{R}((l,s),\vartheta\cup(\chi+t-u),t)$, meaning that state $(l,\vartheta,t)\|_s(s,\chi,u)$ is a reachable state. Using Lemma 5.1.26, we thus obtain that $\mathcal{U}_t((l,\vartheta,t)\|_s(s,\chi,u))$, and therefore, we can derive that $\chi,(\chi+t-u) \models \iota_2(s)$ must hold. Combining this with $(\chi+t-u),(\chi+u'-u) \models \iota_2(s)$ allows us to derive that also $\chi,(\chi+u'-u) \models \iota_2(s)$. In combination with $\vartheta,(\vartheta+u'-t) \models \iota_1(l)$, we thus have $\mathcal{U}_{u'}(l,\vartheta,t)$ and $\mathcal{U}_{u'}(s,\chi,u)$, and therefore also the required $\mathcal{U}_{u'}((l,\vartheta,t)\|_s(s,\chi,u))$.

2. Assume we have a transition $(l,\vartheta,t)\|_s(s,\chi,u) \xrightarrow{\sigma}_{t'} (l',\vartheta',t')\|_s(s,\chi,u)$ for some $\sigma \in \Sigma_1 \setminus \Sigma_2$ and $t' \geq t$. By definition, we must then also have $(l,\vartheta,t) \xrightarrow{\sigma}_{t'} (l',\vartheta',t')$ and $\mathcal{U}_{t'}(s,\chi,u)$. Hence, we know there exists a switch $l \xrightarrow{\sigma,\varphi,\lambda} l'$, such that $\vartheta,(\vartheta+t'-t) \models \iota_1(l)$, $\vartheta' = (\vartheta+t'-t)[\lambda := \mathbf{0}]$, $\vartheta' \models \iota_1(l')$, $(\vartheta+t'-t) \models \varphi$, and $\chi,(\chi+t'-u) \models \iota_2(s)$. The latter clause implies $(\chi+t-u),((\chi+t-u)+t'-t) \models \iota_2(s)$ using the interpolation property 5.1.6. Thus, it immediately follows that we also have a transition $((l,s),\vartheta\cup(\chi+t-u),t) \xrightarrow{\sigma}_{t'} ((l',s),\vartheta'\cup(\chi+t'-u),t')$. By definition, we have $(l',\vartheta',t')\|_s(s,\chi,u)\mathcal{R}((l',s),\vartheta'\cup(\chi+t'-u),t')$, as $t' \geq t \geq u$. The case where $\sigma \in \Sigma_2\setminus\Sigma_1$ is similar to the above case.

   Conversely, assume we have a transition $((l,s),\vartheta\cup(\chi+t-u),t) \xrightarrow{\sigma}_{t'} ((l',s),\vartheta'\cup(\chi+t'-u),t')$ for some $\sigma \in \Sigma_1 \setminus \Sigma_2$ and $t' \geq t$. This means there is a switch $(l,s) \xrightarrow{\sigma,\varphi,\lambda} (l',s)$, for which we have $\vartheta\cup(\chi+t-u),((\vartheta\cup(\chi+t-u))+t'-t) \models \iota(l,s)$, $\vartheta'\cup(\chi+t'-u) = ((\vartheta\cup(\chi+t-u))+t'-t)[\lambda := \mathbf{0}]$, $((\vartheta\cup(\chi+t-u))+t'-t) \models \varphi$ and $\vartheta'\cup(\chi+t'-u) \models \iota(l',s)$. We again use the assumption that $(l,\vartheta,t)\|_s(s,\chi,u)$ is a reachable state, allowing us to use Lemma 5.1.26 to derive that $\chi,(\chi+u'-t) \models \iota_2(s)$. Combining this with the above derived clauses, we can deduce that there must be a switch $l \xrightarrow{\sigma,\varphi,\lambda} l'$, such that $\vartheta,(\vartheta+t'-t) \models \iota_1(l)$, $\vartheta' = (\vartheta+t'-t)[\lambda := \mathbf{0}]$, $\vartheta' \models \iota_1(l)$, $(\vartheta+t'-t) \models \varphi$ and $\chi,(\chi+t'-u) \models \iota_2(s)$. Thus, we also have $(l,\vartheta,t) \xrightarrow{\sigma}_{t'} (l',\vartheta',t')$ and $\mathcal{U}_{t'}(s,\chi,u)$, which is sufficient to derive $(l,\vartheta,t)\|_s(s,\chi,u) \xrightarrow{\sigma}_{t'} (l',\vartheta',t')\|_s(s,\chi,u)$. Obviously, we also have $(l',\vartheta',t')\|_s(s,\chi,u)\mathcal{R}((l',s),\vartheta'\cup(\chi+t'-u),t')$, as $t' \geq t \geq u$. The case where $\sigma \in \Sigma_2 \setminus \Sigma_1$ is again similar to the above case.

3. Assume we have a transition $(l,\vartheta,t)\|_s(s,\chi,u) \xrightarrow{\sigma}_{t'} (l',\vartheta',t')\|_s(s',\chi',t')$ for some $\sigma \in \Sigma_1 \cap \Sigma_2$. By definition, we must then also have $(l,\vartheta,t) \xrightarrow{\sigma}_{t'} (l',\vartheta',t')$ and $(s,\chi,u) \xrightarrow{\sigma}_{t'} (s',\chi',t')$. Thus, there must exist switches $l \xrightarrow{\sigma,\varphi,\lambda} l'$ and $s \xrightarrow{\sigma,\psi,\kappa} s'$, such that $\vartheta,(\vartheta+t'-t) \models \iota_1(l)$, $\chi,(\chi+t'-u) \models \iota_2(s)$, $\vartheta' = (\vartheta+t'-t)[\lambda := \mathbf{0}]$, $\chi' = (\chi+t'-u)[\kappa := \mathbf{0}]$,

$\vartheta' \models \iota_1(l)$, $\chi' \models \iota_2(s)$, $(\vartheta + t' - t) \models \varphi$ and $(\chi + t' - u) \models \psi$. Using the interpolation property 5.1.6, we thus arrive at the existence of a switch $(l, s) \xrightarrow{\sigma,\varphi\wedge\psi,\lambda\cup\kappa} (l', s')$, such that $\vartheta\cup(\chi+t-u), ((\vartheta\cup(\chi+t-u))+t'-t) \models \iota(l, s), \vartheta'\cup\chi' = ((\vartheta\cup(\chi+t-u))+t'-t)[\lambda\cup\kappa := \mathbf{0}]$, $\vartheta' \cup \chi' \models \iota(l', s')$ and $((\vartheta \cup (\chi + t - u)) + t' - t) \models (\varphi \wedge \psi)$, and therefore we have a transition $((l, s), \vartheta \cup (\chi + t - u), t) \xrightarrow{\sigma}_{t'} ((l', s'), \vartheta' \cup \chi', t')$. By definition, we have $(l', \vartheta', t')\|_s(s', \chi', t')\mathcal{R}((l', s'), \vartheta' \cup \chi', t')$.

The converse follows the same steps in reverse order and again employs the fact that $(l, \vartheta, t)\|_s(s, \chi, u)$ is a reachable state.

The proof of the case where $(l, \vartheta, t)\|_s(s, \chi, u)\mathcal{R}((l, s), (\vartheta + u - t) \cup \chi, u)$ and $u \geq t$ is fully symmetric to the above discussed cases. □

5.1.4 Semantics — Unity

The motivation for introducing the alternative semantics in the first place, is to allow for a straightforward interpretation of timed automata in μCRL$_t$. Defining a semantics of a timed automaton using the underlying model of TSLTSs significantly reduces the complexity of this task, as it allows us to focus on the matters of relevance.

Of course, doing this intermediate interpretation only pays-off if by doing so, no information is lost. This implies that the two defined semantics must be tightly related. This relation is subject of investigation in this Section.

We first establish a connection between the delay predicate of the alternative semantics of a timed automaton and the delay transitions of the standard semantics of a timed automaton. In Lemma 5.1.29, we show the correspondence between a time-stamped transition of the alternative semantics and the combination of a delay transitions and action transition of the standard semantics.

Lemma 5.1.28. For all timed automata X and delays $d \geq \mathbf{0}$, Eqn. (5.10) holds.

$$\mathcal{U}_{t+d}(l, \vartheta, t) \ \Leftrightarrow\ (l, \vartheta, t) \overset{t+d}{\longmapsto} (l, \vartheta + d, t + d) \tag{5.10}$$

Proof. Given $d \geq \mathbf{0}$ and $\mathcal{U}_{t+d}(l, \vartheta, t)$, we know $\vartheta, (\vartheta + d) \models \iota(l)$. This immediately allows us to derive the required transition $(l, \vartheta, t) \overset{t+d}{\longmapsto} (l, \vartheta + d, t + d)$. Analogously, the converse can be proved. □

Lemma 5.1.29. For all timed automata X, actions $\sigma \in \Sigma$ and delays $d \geq \mathbf{0}$, Eqn. (5.11) holds.

$$(l, \vartheta, t) \xrightarrow{\sigma}_{t+d} (l', \vartheta', t + d) \ \Leftrightarrow\ (l, \vartheta, t) \overset{t+d}{\longmapsto} (l, \vartheta + d, t + d) \xrightarrow{\sigma} (l', \vartheta', t + d) \tag{5.11}$$

Proof. Observe that for $d \geq \mathbf{0}$, $(l, \vartheta, t) \xrightarrow{\sigma}_{t+d} (l', \vartheta', t + d)$ implies $\mathcal{U}_{t+d}(l, \vartheta, t)$, thus we immediately have the delay transition $(l, \vartheta, t) \overset{t+d}{\longmapsto} (l, \vartheta + d, t + d)$. It then is easily verified that in order to execute the time-stamped transition, all requirements for executing an action transition according to the standard semantics must be fulfilled. Vice versa, these latter requirements, together with $\mathcal{U}_{t+d}(l, \vartheta, t)$ are sufficient to prove the required time-stamped transition. □

The above lemmata are then the basis for the main result, showing that both semantics are essentially identical and differentiate equal sets of automata. Hence, the choice for either semantics is immaterial.

Theorem 5.1.30 For all timed automata X_1 and X_2, Eqn. (5.12) holds.

$$X_1 \,\underline{\leftrightarrow}{}^{\iota}_{t} X_2 \;\Leftrightarrow\; X_1 \,\underline{\leftrightarrow}_{t} X_2 \tag{5.12}$$

Proof. If $\mathcal{R}{:}X_1 \,\underline{\leftrightarrow}{}^{\iota}_{t} X_2$, then also $\mathcal{R}{:}X_1 \,\underline{\leftrightarrow}_{t} X_2$: for $\mathcal{R}$ to be a bisimulation, the conditions with respect to the initial states must be met. Obviously, this is true, since $\mathcal{R}$ already is a timed bisimulation relation for $X_1 \,\underline{\leftrightarrow}{}^{\iota}_{t} X_2$. We proceed by proving the following cases.

1. Suppose $(l, \vartheta, t)\mathcal{R}(s, \chi, t)$ and suppose $\mathcal{U}_{t'}(l, \vartheta, t)$. If $t' < t$, then there must be a $t'' \geq t$ such that $\mathcal{U}_{t''}(l, \vartheta, t)$; hence, it suffices to verify the case where $t' \geq t$. Then, using Lemma 5.1.28, we know $(l, \vartheta, t) \xmapsto{t'} (l, \vartheta + t' - t, t')$ and since $\mathcal{R}$ is a timed bisimulation relation, we also have $(s, \chi, t) \xmapsto{t'} (s, \chi + t' - t, t')$. Moreover, we also have $(l, \vartheta + t' - t, t')\mathcal{R}(s, \chi + t' - t, t')$.

2. Suppose $(l, \vartheta, t)\mathcal{R}(s, \chi, t)$ and suppose $(l, \vartheta, t) \xrightarrow{\sigma}_{t'} (l', \vartheta', t')$. By Lemma 5.1.29, we then have $(l, \vartheta, t) \xmapsto{t'} (l, \vartheta + t' - t, t')$ and $(l, \vartheta + t' - t, t') \xrightarrow{\sigma} (l', \vartheta', t')$. Since we know $\mathcal{R}$ is a timed bisimulation, relating X_1 and X_2 under TPLTS semantics, we know there exist states $(s, \chi + t' - t, t')$ and (s', χ', t'), such that $(s, \chi, t) \xmapsto{t'} (s, \chi + t' - t, t')$ and $(s, \chi + t' - t, t') \xrightarrow{\sigma} (s', \chi', t')$. Using Lemma 5.1.29 again yields $(s, \chi, t) \xrightarrow{\sigma}_{t'} (s', \chi', t')$. Moreover, we have $(l', \chi', t')\mathcal{R}(s', \chi', t')$ and therefore, also $(l', \chi', t')\mathcal{R}'(s', \chi', t')$.

Conversely, suppose we have a relation $\mathcal{R}$, such that $\mathcal{R}{:}X_1 \,\underline{\leftrightarrow}_{t} X_2$; then we can define a relation $\mathcal{R}'$, such that $\mathcal{R}'{:}X_1 \,\underline{\leftrightarrow}{}^{\iota}_{t} X_2$. Let $\mathcal{R}'$ be as defined by Eqn. (5.13).

$$\mathcal{R}' = \mathcal{R} \cup \{\langle (l, \vartheta + d, t + d), (s, \chi + d, t + d)\rangle \mid (l, \vartheta, t)\mathcal{R}(s, \chi, t) \wedge \mathcal{U}_{t+d}(l, \vartheta, t) \wedge \mathbf{0} \leq d\} \tag{5.13}$$

Again, for the initial states, it is not hard to see $\mathcal{R}'$ satisfies the requirements of timed bisimilarity. We distinguish the following two cases.

1. Suppose $(l, \vartheta, t)\mathcal{R}'(s, \chi, t)$ and assume $(l, \vartheta, t) \xmapsto{t+d} (l, \vartheta + d, t + d)$. By Lemma 5.1.28, we then have $\mathcal{U}_{t+d}(l, \vartheta, t)$. Since $\mathcal{R}$ is a timed bisimulation relation, we have $\mathcal{U}_{t+d}(s, \chi, t)$ as well. Therefore, also $(s, \chi, t) \xmapsto{t+d} (s, \chi + d, t + d)$. By construction, we then have $(l, \vartheta + d, t + d)\mathcal{R}'(s, \chi + d, t + d)$.

2. Suppose $(l, \vartheta, t)\mathcal{R}'(s, \chi, t)$ and assume $(l, \vartheta, t) \xrightarrow{\sigma} (l', \vartheta', t)$. By Lemma 5.1.29, we then have $(l, \vartheta, t) \xrightarrow{\sigma}_{t} (l', \vartheta', t)$. Since $\mathcal{R}$ is a timed bisimulation relation, we have $(s, \chi, t) \xrightarrow{\sigma}_{t} (s', \chi', t)$, and, moreover, $(l', \vartheta', t)\mathcal{R}(s', \chi', t)$. Thus, $(s, \chi, t) \xrightarrow{\sigma} (s', \chi, t)$ and $(l', \vartheta', t)\mathcal{R}'(s', \chi', t)$

$\square$

5.2 Hybrid Automata

Hybrid Automata have first been proposed in [6]. By many computer scientists, they are judged as the answer to many of the gnawing questions that are raised in the complex field of hybrid systems. Hybrid Systems are characterised by the mixture, and interplay, of two types of events, viz. discrete events and continuous events. There is an abundance of theory describing how to deal with systems harbouring only one of these types of events. It is, however, in their combined presence that questions remain largely unanswered. Typical examples of hybrid systems are e.g. digital controllers for analog components or plants.

A hybrid automaton is a formal model for such a mixed discrete-continuous system. Hybrid Automata can be seen as a generalisation of timed automata: the real-time clocks of a timed automaton are constrained to progress at a fixed rate; in hybrid automata more complex progressions are allowed.

In section 5.2.1, we introduce the syntax of hybrid automata. Section 5.2.2 then introduces (an absolute time version of) the standard semantics of a hybrid automaton. An alternative to this semantics is provided in section 5.2.3. Section 5.2.4 finally investigates the relation between the standard semantics and the proposed alternative semantics.

5.2.1 Syntax

Similar to discrete systems, where an alphabet of distinct symbols is used to model discrete events, continuous systems are modelled using an alphabet of distinct symbols, modelling *continuous events*. We often refer to a symbol representing a continuous event as a *continuous variable*. These continuous variables capture (or should capture) the essential dynamics of the system by means of constraints on the values these variables (and their derivatives) can obtain, e.g. by means of inequalities between constants and variables.

Definition 5.2.1 (*Continuous Variables*).
A continuous variable is a variable that can assume any value in the set of real numbers and is thought to change continuously in time. Henceforth, we will assume an infinite set of continuous variables $\mathcal{V}$. Typical continuous variables are x, x_1, x_2, etc. To each variable x in $\mathcal{V}$, we associate a first derivative $\dot{x}$. The set $\dot{\mathcal{V}} = \{\dot{x} \mid x \in \mathcal{V}\}$, consists of these first derivatives. Finally, the set $\mathcal{V}'$ is defined as the set $\{x' \mid x \in \mathcal{V}\}$, and is used to denote the set of continuous variables after substitution according to a given predicate.

Definition 5.2.2 (*Snapshot Valuation*, *Valuation*).
A snapshot valuation of continuous variables assigns a real value to the continuous variables, i.e. a snapshot valuation is a mapping $\vartheta{:}\mathcal{V} \cup \dot{\mathcal{V}} \to \mathbb{R}^{\perp}$, such that for every variable $x \in \mathcal{V} \cup \dot{\mathcal{V}}$, $\vartheta(x) \in \mathbb{R}$ if it is defined and $\vartheta(x) = \perp$ otherwise. The set of all snapshot valuations is denoted by $\mathbb{V}$. Since we are often interested only in a (finite) subset of continuous variables, we introduce the notation of $\mathbb{V}_V$ for a set $V \subseteq \mathcal{V}$, denoting the set of all snapshot valuations $\vartheta{:}V \cup \dot{V} \to \mathbb{R}^{\perp}$. Similarly, we have snapshot valuations for variables of the set $\mathcal{V}'$. The set of all snapshot valuations $\vartheta'{:}\mathcal{V}' \to \mathbb{R}^{\perp}$ is denoted $\mathbb{V}'$, whereas the set $\mathbb{V}'_V$ denotes the set of all snapshot valuations $\vartheta'{:}V' \to \mathbb{R}^{\perp}$ for all subsets $V \subseteq \mathcal{V}$.
A valuation of continuous variables is a (partial) mapping f of elements of the time-line $\mathbb{R}_{\geq \mathbf{0}}$ to the set of snapshot valuations $\mathbb{V}$, i.e. $f{:}\mathbb{R}_{\geq \mathbf{0}} \to \mathbb{V}$. The set of all valuations is denoted by $\mathbb{F}$. Likewise, we use $\mathbb{F}_V$ to denote the set of all valuations $f{:}\mathbb{R}_{\geq \mathbf{0}} \to \mathbb{V}_V$.

Definition 5.2.3 (*Constraint*).
The relation between continuous variables, their derivatives and constants is expressed as a *constraint*. The grammar, defined in Eqn. (5.14) specifies the syntax of the constraints.

$$\gamma ::= \mathsf{t} \mid \mathsf{f} \mid r(\chi_1, \ldots, \chi_n) \mid \neg\gamma \mid \gamma_1 \wedge \gamma_2 \mid \gamma_1 \vee \gamma_2 \tag{5.14}$$

Here, $r{:}\mathbb{R}^n \to \mathbb{B}$ is a relation and the expressions $\chi_1, \ldots, \chi_n$ represent elements of the real numbers. The data-grammar specifying the allowed expressions χ_i depends on the type of constraint and is discussed elsewhere in this section. The interpretation $[\![\gamma]\!]^\vartheta$ of a constraint γ under a snapshot valuation ϑ is defined inductively by (5.15).

$$\begin{aligned}
[\![\mathsf{t}]\!]^\vartheta &= \mathsf{t}\\
[\![\mathsf{f}]\!]^\vartheta &= \mathsf{f}\\
[\![\neg\gamma]\!]^\vartheta &= \neg[\![\gamma]\!]^\vartheta\\
[\![\gamma_1 \wedge \gamma_2]\!]^\vartheta &= [\![\gamma_1]\!]^\vartheta \wedge [\![\gamma_2]\!]^\vartheta\\
[\![\gamma_1 \vee \gamma_2]\!]^\vartheta &= [\![\gamma_1]\!]^\vartheta \vee [\![\gamma_2]\!]^\vartheta\\
[\![r(\chi_1, \ldots, \chi_n)]\!]^\vartheta &= r([\![\chi_1]\!]^\vartheta, \ldots, [\![\chi_n]\!]^\vartheta)
\end{aligned} \tag{5.15}$$

Definition 5.2.4 (*Variable Constraints*).
A variable constraint is a relation between a set of continuous variables and the real numbers. For the set $\mathcal{V}$ of continuous variables, the set of variable constraints $\Phi(\mathcal{V})$ is the set of constraints, generated by the grammar (5.14) where the data-grammar is defined by (5.16).

$$\chi ::= c \mid x \mid F(\chi_1, \ldots, \chi_n) \tag{5.16}$$

Here, $c \in \mathbb{R}$, $x \in \mathcal{V}$ and $F{:}\mathbb{R}^n \to \mathbb{R}$.

The interpretation $[\![\varphi]\!]^\vartheta$ of a variable constraint $\varphi \in \Phi(\mathcal{V})$ under a snapshot valuation ϑ is defined by Eqn. (5.15), where the interpretation of the data-grammar is defined in Eqn. (5.17).

$$[\![F(\chi_1, \ldots, \chi_n)]\!]^\vartheta \quad = F([\![\chi_1]\!]^\vartheta, \ldots, [\![\chi_n]\!]^\vartheta) \tag{5.17}$$

Note that for constants c we define $[\![c]\!]^\vartheta = c$ and for continuous variables x we define $[\![x]\!]^\vartheta = \vartheta(x)$. Satisfaction of a variable constraint φ in a snapshot valuation ϑ, denoted by $\vartheta \models \varphi$ means that the closed formula $[\![\varphi]\!]^\vartheta$ holds. If a variable constraint φ is not satisfied under snapshot valuation φ, we write $\vartheta \not\models \varphi$. We say a variable constraint φ is *satisfiable* iff there exists a valuation ϑ, such that $\vartheta \models \varphi$. The interpretation of a variable constraint $\varphi \in \Phi(\mathcal{V})$ under a valuation f is defined as the function $[\![\varphi]\!]^f = \lambda\varepsilon{:}\mathsf{dom}(f).\ [\![\varphi]\!]^{f(\varepsilon)}$. Satisfaction of a variable constraint φ in a valuation f, denoted by $f \models \varphi$ means that the closed formula $\forall_{\varepsilon:\mathsf{dom}(f)}\ [\![\varphi]\!]^{f(\varepsilon)}$ holds.Whenever a variable constraint φ is not satisfied under valuation f, we write $f \not\models \varphi$.

Definition 5.2.5 (*Flow Constraint*).
A flow constraint is a constraint on the set of continuous variables and their derivatives. For the set $\mathcal{V} \cup \dot{\mathcal{V}}$ of continuous variables and their first-derivatives, the set $\Theta(\mathcal{V} \cup \dot{\mathcal{V}})$ is the set of constraints, generated by the grammar (5.14) where the data-grammar is defined by (5.18).

$$\chi \quad ::= c \mid x \mid \dot{x} \mid F(\chi_1, \ldots, \chi_n) \tag{5.18}$$

Here, $c \in \mathbb{R}$, $x \in \mathcal{V}$, $\dot{x} \in \dot{\mathcal{V}}$ and $F{:}\mathbb{R}^n \to \mathbb{R}$.

The interpretation of a flow constraint $\theta \in \Theta(\mathcal{V} \cup \dot{\mathcal{V}})$ under a snapshot valuation ϑ is defined inductively by Eqn. (5.15), where the data-grammar is interpreted in Eqn. (5.19).

$$[\![F(\chi_1, \dots, \chi_n)]\!]^{\vartheta} \quad = F([\![\chi_1]\!]^{\vartheta}, \dots, [\![\chi_n]\!]^{\vartheta}) \tag{5.19}$$

Note that for constants c, we define $[\![c]\!]^{\vartheta} = c$ and for continuous variables x with their first derivatives $\dot{x}$, we define $[\![x]\!]^{\vartheta} = \vartheta(x)$ and $[\![\dot{x}]\!]^{\vartheta} = \vartheta(\dot{x})$. The interpretation $[\![\theta]\!]^{\vartheta}$ of a flow constraint $\theta \in \Theta(\mathcal{V} \cup \dot{\mathcal{V}})$ under a valuation f is defined as the function $[\![\theta]\!]^{f} = \lambda\varepsilon{:}\mathsf{dom}(f).\ [\![\theta]\!]^{f\varepsilon}$. Satisfaction of a flow constraint θ in a valuation f, denoted by $f \models \theta$ means that the closed formula $\forall_{\varepsilon:\mathsf{dom}(f)}\ [\![\theta]\!]^{f(\varepsilon)}$ holds. Whenever a flow constraint θ is not satisfied under a valuation f, we write $f \not\models \theta$. Note that, by definition, for all f, such that $\mathsf{dom}(f) = \emptyset$, and all θ we have $f \models \theta$.

Definition 5.2.6 (*Reset Constraints*).
A reset constraint is a constraint on the set of continuous variables and their updated counterparts. The set of reset constraints $R(\mathcal{V} \cup \mathcal{V}')$ is the set of constraints, generated by the grammar (5.14), where the data-grammar is defined by (5.20).

$$\chi ::= c \mid x \mid x' \mid F(\chi_1, \dots, \chi_n) \tag{5.20}$$

Here, $c \in \mathbb{R}$, $x \in \mathcal{V}$, $x' \in \mathcal{V}'$ and $F{:}\mathbb{R}^n \to \mathbb{R}$.

The interpretation $[\![\rho]\!]^{\vartheta,\vartheta'}$ of a reset constraint $\rho \in R(\mathcal{V} \cup \mathcal{V}')$ under snapshot valuations ϑ and ϑ', where $\vartheta{:}\mathcal{V} \to \mathbb{R}^{\perp}$ and $\vartheta'{:}\mathcal{V}' \to \mathbb{R}^{\perp}$ is defined inductively by Eqn. (5.15), where the interpretation of the data-grammar is defined by Eqn. (5.21).

$$[\![F(\chi_1, \dots, \chi_n)]\!]^{\vartheta,\vartheta'} \quad = F([\![\chi_1]\!]^{\vartheta,\vartheta'}, \dots, [\![\chi_n]\!]^{\vartheta,\vartheta'}) \tag{5.21}$$

Note that for constants c we define $[\![c]\!]^{\vartheta,\vartheta'} = c$ and for continuous variables x and their counterparts x', we define $[\![x]\!]^{\vartheta,\vartheta'} = \vartheta(x)$ and $[\![x']\!]^{\vartheta,\vartheta'} = \vartheta'(x')$. Satisfaction of a reset constraint ρ in snapshot valuations ϑ, ϑ', denoted by $\vartheta' \models \rho[\vartheta]$ means that the closed formula $[\![\rho]\!]^{\vartheta,\vartheta'}$ holds. If a reset constraint ρ is not satisfied under snapshot valuations ϑ and ϑ', we write $\vartheta' \not\models \rho[\vartheta]$.

Example 5.2.7. Let x, y be continuous variables. The type of constraints we can specify are e.g.

- variable constraints: $x \leq y \wedge 4 \leq x^2$,
- flow constraints: $\dot{x} = y \wedge \dot{y} = 10$,
- reset constraints: $0 \leq x' \leq 5 \wedge y' = y$.

The flow constraint example shows we can use higher order derivatives of continuous variables, since we can also derive $\ddot{x} = 10$.

We next present the textual syntax of a hybrid automaton. Like timed automata, hybrid automata can also be depicted graphically; we shall, however, not go into the formal definition of its graphical syntax, but limit its explanation to an informal exposition.

Definition 5.2.8 (*Hybrid Automaton*).
Formally, a hybrid automaton X is a tuple $\langle M, M^0, \Sigma, V, \iota, \theta, \mathcal{I}, E\rangle$,where

- M is a finite set of *control modes*,
- $\emptyset \subset M^0 \subseteq M$ is a set of *initial* control modes,
- Σ is a finite set of labels,
- $V \subseteq \mathcal{V}$ is a finite set of continuous variables,
- $\iota{:}M \to \Phi(V)$ is a mapping that labels each location m in M with some variable constraint in $\Phi(V)$,
- $\theta{:}M \to \Theta(V \cup \dot{V})$ is the flow condition function,
- $\mathcal{I}{:}M^0 \to \Phi(V)$ is a mapping that assigns to each initial control mode an initialisation constraint in $\Phi(V)$. We require that for all initial modes m and snapshot valuations ϑ, $[\![\mathcal{I}(m)]\!]^{\vartheta} \Rightarrow [\![\iota(m)]\!]^{\vartheta}$,
- $E \subseteq M \times \Sigma \times \Phi(V) \times R(V \cup V') \times M$ is a set of *switches*.

We write $m \xrightarrow{\sigma,\varphi,\rho} m'$ rather than $(m, \sigma, \varphi, \rho, m') \in E$.

Remark 5.2.9. *Restricting the initialisation constraint $\mathcal{I}$ to constraints that are satisfiable in conjunction with the control mode invariant is needed for ensuring the hybrid automaton can at least be initialised; the requirement is comparable to the one for timed automata (see Remark 5.1.8).*

A graphical notation for hybrid automata, consists of a directed graph, connecting nodes via edges. Like timed automata, the graph is annotated with additional information. The nodes, representing the control modes of the automaton, carry the information contained in the invariant and the flow condition functions. The edges, representing the switches, are annotated with the action, a guard and a reset constraint.

Example 5.2.10. A typical example of a hybrid system is a controller for a tank containing water. The controller ensures the water level in the tank remains between B and T ($B < T$) litres, in the presence of a continuous drain of D litres per second. It does this by opening (resp. closing) a faucet, which causes water at I litres per second to flow into the tank ($I \geq D$). The model for the controller is depicted in Fig. 5.6.

Using a parallel composition operator $\|_s$, hybrid automata allow for the compositional modelling of complex hybrid systems. The composition mechanism is essentially the same as for timed automata. Note that there exist variations on the concept of hybrid automata that allow for other communications primitives, e.g. *Hybrid I/O Automata* [85], which can also use communication over shared continuous variables. Here, we restrict our attention to *synchronising hybrid automata*, consisting of the parallel composition of hybrid automata using the operator $\|_s$.

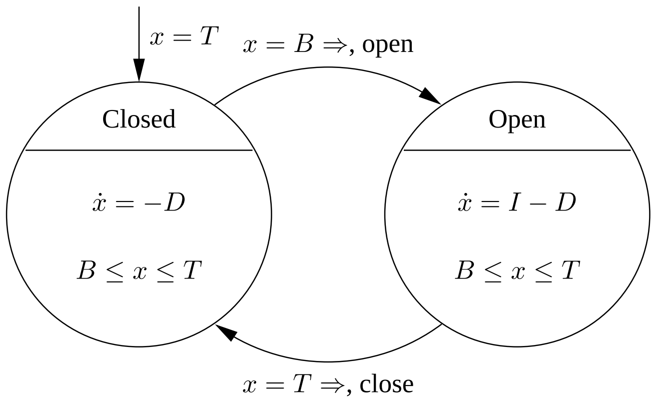


Figure 5.6: Water-Level Controller

Definition 5.2.11 (*Synchronising Hybrid Automata*).
Let $X_1 = \langle M_1, M_1^0, \Sigma_1, V_1, \iota_1, \theta_1, \mathcal{I}_1, E_1 \rangle$ and $X_2 = \langle M_2, M_2^0, \Sigma_2, V_2, \iota_2, \theta_2, \mathcal{I}_2, E_2 \rangle$ be hybrid automata, for which their sets of continuous variables are disjoint, i.e. $V_1 \cap V_2 = \emptyset$. The parallel composition of X_1 and X_2, denoted by $X_1 \|_s X_2$ is the hybrid automaton, defined by the tuple $\langle M_1 \times M_2, M_1^0 \times M_2^0, \Sigma_1 \cup \Sigma_2, V_1 \cup V_2, \iota, \theta, \mathcal{I}, E \rangle$, where the variable constraints ι and $\mathcal{I}$ are defined as $\iota(m,n) = \iota_1(m) \wedge \iota_2(n)$ and $\mathcal{I}(m,n) = \mathcal{I}_1(m) \wedge \mathcal{I}_2(n)$, and the flow constraint θ is defined as $\theta(m,n) = \theta_1(m) \wedge \theta_2(n)$. The set of switches E of the composite automaton is the least set defined according to the rules of Table 5.6.

$$\frac{m \xrightarrow{\sigma,\varphi,\rho} m' \quad n \xrightarrow{\sigma,\psi,\pi} n'}{(m,n) \xrightarrow{\sigma,\varphi\wedge\psi,\rho\wedge\pi} (m',n')} \qquad \frac{m \xrightarrow{\sigma,\varphi,\rho} m' \quad \sigma \in \Sigma_1 \setminus \Sigma_2}{(m,n) \xrightarrow{\sigma,\varphi,\rho} (m',n)} \qquad \frac{n \xrightarrow{\sigma,\psi,\pi} n' \quad \sigma \in \Sigma_2 \setminus \Sigma_1}{(m,n) \xrightarrow{\sigma,\psi,\pi} (m,n')}$$

Table 5.6: Deduction Rules for Synchronisation

5.2.2 Semantics — Standard

The standard interpretation, or semantics, of a hybrid automaton is defined using the underlying model of a TPLTS. In this section, we repeat the standard semantics based on absolute time rather than relative time, as we also did for timed automata. Before we define the semantics, we introduce additional notation.

Notation 5.2.12. We write $f \rceil R$ for the restriction of function f to the domain of f coinciding with R, i.e. $\mathsf{dom}(f \rceil R) = \mathsf{dom}(f) \cap R$ and $\forall_{\varepsilon \in \mathsf{dom}(f \rceil R)} \; (f \rceil R)\varepsilon = f\varepsilon$. For a given valuation $f \in \mathbb{F}$, we write $\mathsf{diff}(f)$ if f is strictly differentiable on its domain and we write $\mathsf{cont}(f)$ if f is strictly continuous on its domain.

Definition 5.2.13 (*Semantics of a Hybrid Automaton (TPLTS)*).
Let $X = \langle M, M^0, \Sigma, V, \iota, \theta, \mathcal{I}, E \rangle$ be a hybrid automaton. The semantics of X is given by the TPLTS $X^s = \langle M \times \mathbb{V}_V \times \mathbb{R}_{\geq \mathbf{0}}, (M^0 \times \mathbb{V}_V \times \{\mathbf{0}\})_{\mathcal{I}}, \Sigma, \rightarrow, \mapsto \rangle$. Note that we use the notation

$(M^0 \times \mathbb{V}_V \times \{\mathbf{0}\})_{\mathcal{I}}$ to denote the set of elements $(m, \vartheta, \mathbf{0}) \in M^0 \times \mathbb{V}_V \times \{\mathbf{0}\}$ satisfying $\vartheta \models \mathcal{I}(m)$ for $m \in M^0$. The action transition $\rightarrow$ and the delay transition $\mapsto$ are defined as the least relations satisfying the rules of Table 5.7.

$$\frac{m \xrightarrow{\sigma,\varphi,\rho} m' \quad \vartheta \models \iota(m) \quad \vartheta \models \varphi \quad \vartheta' \models \rho[\vartheta] \quad \vartheta' \models \iota(m')}{(m, \vartheta, t) \xrightarrow{\sigma} (m', \vartheta', t)}$$

$$\frac{\begin{array}{c} f{:}[0,d] \rightarrow \mathbb{V}_V \quad f(\mathbf{0}) = \vartheta \quad f(d) = \vartheta' \quad \mathsf{cont}(f) \\ \mathsf{diff}(f\rceil(0,d)) \quad f \models \iota(m) \quad f\rceil(0,d) \models \theta(m) \quad \mathbf{0} \leq d \end{array}}{(m, \vartheta, t) \overset{t+d}{\longmapsto} (m, \vartheta', t+d)}$$

Table 5.7: Rules for the Two-Phase Semantics

The state-space of X^s consists of the set of states that can be reached from the initial states

Remark 5.2.14. *The existence of a continuous, differentiable function (often referred to as a witness) is sufficient for delaying a certain amount of time. The witness, however, is not part of the states or transitions of the resulting two-phase transition system. Moreover, there is no requirement on the uniqueness of the witness for a transition. In some sense, the semantics of a hybrid automaton abstracts from the continuous behaviour, and therefore might not adequately model hybrid systems. Notice that in Hybrid I/O Automata [85, 86, 87], the witness is part of the transition relation.*

Definition 5.2.15 (*Timed Bisimulation for Hybrid Automata (TPLTS)*).
Let X_1 and X_2 be two hybrid automata, and let X_1^s and X_2^s be the interpretations of X_1 and X_2 in terms of TPLTSs. A relation $\mathcal{R} \subseteq (M_1 \times \mathbb{V}_{V_1} \times \mathbb{R}_{\geq \mathbf{0}}) \times (M_2 \times \mathbb{V}_{V_2} \times \mathbb{R}_{\geq \mathbf{0}})$ is a timed bisimulation iff for all states (m_1, ϑ, t) and (m_2, χ, u), the following properties are satisfied.

1. if $(m_1, \vartheta, t)\mathcal{R}(m_2, \chi, u)$ and $(m_1, \vartheta, t) \xrightarrow{\sigma} (m_1', \vartheta', t)$ for some $\sigma \in \Sigma_1$, then there exists a state (m_2', χ', u), such that $(m_2, \chi, u) \xrightarrow{\sigma} (m_2', \chi', u)$ and $(m_1', \vartheta', t)\mathcal{R}(m_2', \chi', u)$. Similarly for each action transition of automaton X_2.

2. if $(m_1, \vartheta, t)\mathcal{R}(m_2, \chi, u)$ and $(m_1, \vartheta, t) \overset{t+d}{\longmapsto} (m_1, \vartheta', t+d)$ for some $d \in \mathbb{R}_{\geq \mathbf{0}}$, then there exists a state $(m_2, \chi', u+d)$, such that $(m_2, \chi, u) \overset{u+d}{\longmapsto} (m_2, \chi', u+d)$ and $(m_1, \vartheta', t+d)\mathcal{R}(m_2, \chi', u+d)$. Similarly for each delay transition of automaton X_2.

3. For each state $(m_1, \vartheta, \mathbf{0}) \in (M_1^0 \times \mathbb{V}_{V_1} \times \{\mathbf{0}\})_{\mathcal{I}_1}$ a state $(m_2, \chi, \mathbf{0}) \in (M_2^0 \times \mathbb{V}_{V_2} \times \{\mathbf{0}\})_{\mathcal{I}_2}$ can be identified, such that $(m_1, \vartheta, \mathbf{0})\mathcal{R}(m_2, \chi, \mathbf{0})$, and conversely for each initial state of X_2 a related state of X_1 can be identified.

If there exists a relation $\mathcal{R}$, relating all (reachable) states between X_1^s and X_2^s according to the above-listed criteria, we say the two hybrid automata X_1 and X_2 are timed bisimilar, denoted by $X_1 \underline{\leftrightarrow}{}_t^{\iota} X_2$.

Remark 5.2.16. *Hybrid Automata allowing Zeno behaviour are generally considered flawed models of reality. However, Zeno behaviour can easily arise as a side-effect of abstracting from uninteresting details; moreover in disciplines, other than computer science, Zeno behaviour is dealt with in one way or the other. In [37], Cuijpers and Reniers argued that timed bisimulation is a too weak equivalence for comparing hybrid systems, as it cannot equate behaviour of systems beyond zeno-points. To mend this flaw, Cuijpers and Reniers introduce a novel bisimulation relation, called topological bisimulation. Even though it may turn out to be desirable to use topological bisimulation as the equivalence for hybrid automata, we refrain from doing so in this thesis. This choice is mainly influenced by our chief interest in the relation between μCRL$_t$ and hybrid automata (studied in the next chapter). Since the equivalence of μCRL$_t$ is based on timed bisimulation, it is only natural to use compare μCRL$_t$ and hybrid automata on this level. In any event, adopting topological bisimulation as the underlying equivalence relation in μCRL$_t$ would require a vast amount of new research, as, for example, recursive specifications such as $X = a \cdot X \cdot b$, would no longer have the by now generally accepted solution a^ω.*

Lemma 5.2.17.
Let X_1 and X_2 be two timed bisimilar hybrid automata and $\mathcal{R}$ be a relation relating X_1 and X_2, i.e. $\mathcal{R}{:}X_1 \leftrightarroweq{}^{\iota}_{t} X_2$. Then, for all states (m,ϑ,t) of X_1^s and (n,χ,u) of X_2^s, reachable from the initial states of X_1 and X_2, implication (5.22) holds.

$$(m,\vartheta,t)\mathcal{R}(n,\chi,u) \;\Rightarrow\; u = t \tag{5.22}$$

Proof. See the proof of Lemma 5.1.16. □

For a semantics of synchronising hybrid automata, we adapt the semantics of synchronising timed automata, as defined by Def. 5.1.18.

Definition 5.2.18 (*Semantics of Synchronising Hybrid Automata (TPLTS)*).
Let $X_1 = \langle M_1, M_1^0, \Sigma_1, V_1, \iota_1, \theta_1, \mathcal{I}_1, E_1\rangle$ and $X_2 = \langle M_2, M_2^0, \Sigma_2, V_2, \iota_2, \theta_2, \mathcal{I}_2, E_2\rangle$ be hybrid automata. The semantics of $X_1 \|_s X_2$ is given by the TPLTS $(X_1)^s \|_s (X_2)^s$, defined as $\langle (M_1 \times \mathbb{V}_{V_1} \times \mathbb{R}_{\geq \mathbf{0}}) \times (M_2 \times \mathbb{V}_{V_2} \times \mathbb{R}_{\geq \mathbf{0}}), (M_1^0 \times \mathbb{V}_{V_1} \times \{\mathbf{0}\})_{\mathcal{I}_1} \times (M_2^0 \times \mathbb{V}_{V_2} \times \{\mathbf{0}\})_{\mathcal{I}_2}, \Sigma_1 \cup \Sigma_2, \rightarrow, \mapsto\rangle$, where for all states $(m,\vartheta,t) \in X_1^s$ and $(n,\chi,u) \in X_2^s$, the set of action and delay transitions $\rightarrow$ and $\mapsto$ are defined as the least relations satisfying the rules of Table 5.8.

The rules for constructing a new hybrid automaton, based on the parallel composition of two hybrid automata, described in Def. 5.2.11 are in accordance with the semantic interpretation presented in Def. 5.2.18.

Lemma 5.2.19. Let X_1 and X_2 be two hybrid automata, and let X_1^s and X_2^s be the TPLTS associated to X_1 and X_2, then there exists a timed bisimulation relation $\mathcal{R}$, such that (5.23) holds (see also Fig. 5.7).

$$\mathcal{R} : X_1^s \|_s X_2^s \leftrightarroweq{}^{\iota}_{t} (X_1 \|_s X_2)^s \tag{5.23}$$

Proof. The same line of reasoning applies as for Lemma 5.1.19. □

$$\frac{(m,\vartheta,t) \overset{t+d}{\longmapsto} (m,\vartheta',t+d) \quad (n,\chi,t) \overset{t+d}{\longmapsto} (n,\chi',t+d)}{(m,\vartheta,t)\|_s(n,\chi,t) \overset{t+d}{\longmapsto} (m,\vartheta',t+d)\|_s(n,\chi',t+d)}$$

$$\frac{(m,\vartheta,t) \xrightarrow{\sigma} (m',\vartheta',t) \quad (n,\chi,t) \overset{t}{\longmapsto} (n,\chi,t) \quad \sigma\in\Sigma_1\setminus\Sigma_2}{(m,\vartheta,t)\|_s(n,\chi,t) \xrightarrow{\sigma} (m',\vartheta',t)\|_s(n,\chi,t)}$$

$$\frac{(n,\chi,u) \xrightarrow{\sigma} (n',\chi',u) \quad (m,\vartheta,u) \overset{u}{\longmapsto} (m,\vartheta,u) \quad \sigma\in\Sigma_2\setminus\Sigma_1}{(m,\vartheta,u)\|_s(n,\chi,u) \xrightarrow{\sigma} (m,\vartheta,u)\|_s(n',\chi',u)}$$

$$\frac{(m,\vartheta,t) \xrightarrow{\sigma} (m',\vartheta',t) \quad (n,\chi,t) \xrightarrow{\sigma} (n',\chi',t) \quad \sigma\in\Sigma_1\cap\Sigma_2}{(m,\vartheta,t)\|_s(n,\chi,t) \xrightarrow{\sigma} (m',\vartheta',t)\|_s(n',\chi',t)}$$

Table 5.8: Rules for the Synchronisation Operator

$$\begin{tikzcd}
X_1 \|_s X_2 \arrow[r, dash] \arrow[d, dash] & (X_1 \|_s X_2)^s \\
(X_1)^s \|_s (X_2)^s \arrow[ur, leftrightarrow, shift left, "\iota"] \arrow[ur, dash, shift right, "\tau"'] &
\end{tikzcd}$$

Figure 5.7: Soundness of Synchronising Hybrid Automata

5.2.3 Semantics — Alternative

In this section, we develop an alternative semantics for a hybrid automaton. This alternative semantics is based on the underlying model of a TSLTS.

Judging the definition of the alternative semantics for a timed automaton, the similarity between the standard semantics and the alternative semantics is both striking and intuitive. For hybrid automata, however, this is too much to hope for. The basis for the difficulties to be discussed is the requirements that classify a function as a witness of a delay transition (recall Remark 5.2.14). We first present an example that clearly illustrates the fundamental difficulties we are faced with in defining the alternative semantics.

Example 5.2.20. We consider the behaviour of a thermostat located in a small room. The thermostat switches off if the temperature in the room exceeds $20°$ Celsius. The temperature rise

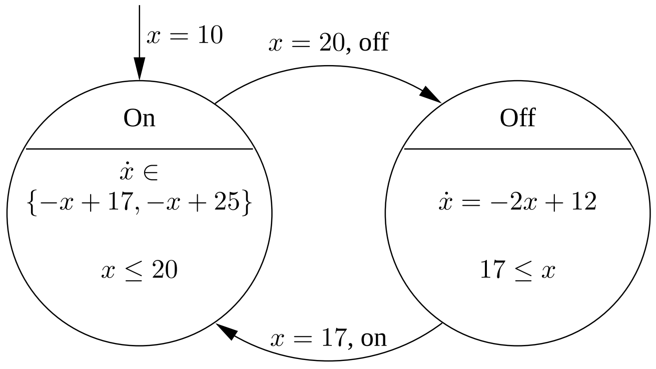


Figure 5.8: Thermostat

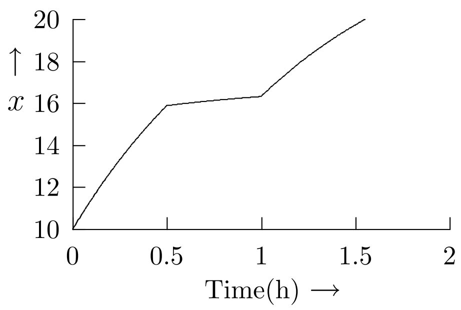


Figure 5.9: Temperature

is non-uniform due to an unpredictable draught, that every now-and-then causes a decrease of the temperature in the room. If the draught persists for some time, it causes the maximal room temperature to fall back to a mere $17°$ Celsius. The process is modelled by the hybrid automaton depicted in Fig. 5.8.

As an effect of the draught, the temperature fluctuates essentially unpredictably when time passes. Figure 5.9, shows only a possible evolution of the temperature in the room. If we zoom in on Fig. 5.9 —see e.g. Fig. 5.10— it is obvious that this particular evolution itself cannot be represented by a single differentiable function, but is necessarily composed of the witnesses of at least three delay transitions in the standard semantics. This means that the event *off* can only be executed in this particular evolution after the automaton has delayed at least three times, visiting intermediate states at time 0.5 and 1.0.

The difficulty that is identified in Example 5.2.20 can be rephrased to the analytic argument that the composition of two continuous and differentiable functions does not necessarily yield a new differentiable function, even if this function is continuous. Hence, the requirement in the delay transition that the witness should be strictly differentiable on its (open) domain is a too strong requirement for a semantics based on a TSLTS, as the delay predicate $\mathcal{U}$ of such a semantics abstracts from intermediate states.

A straightforward transformation of the standard semantics to a semantics based on TSLTSs seems unlikely. The solution is to allow witnesses that are differentiable *almost everywhere*. We

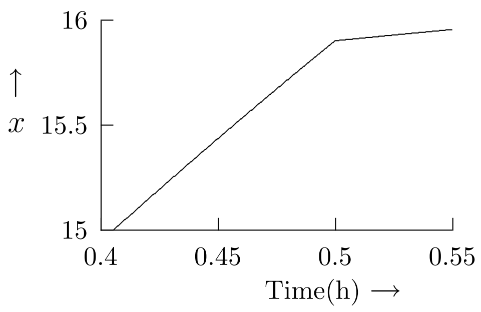


Figure 5.10: Detailed glance at Fig. 5.9

use the Lebesgue measure to define this notion of differentiability. Since we need to borrow a number of definitions and results from measure theory, we first repeat a fraction thereof (for a more in-depth overview see e.g. [61]).

Definition 5.2.21 (*Measure*).
Let X be a set and $\mathfrak{A}$ an algebra on X (i.e. for all $A, B \in \mathfrak{A}$ we have $A \cup B \in \mathfrak{A}$ and for all $A \in \mathfrak{A}$, $A^c \in \mathfrak{A}$, i.e. the complement of A is also in the algebra). A function $\mu{:}\mathfrak{A} \rightarrow \mathbb{R}_{\geq 0} \cup \{\infty\}$ is called a measure if the following properties are satisfied.

1. $\mu(\emptyset) = 0$,
2. $\mu(A) \geq 0$ for all $A \in \mathfrak{A}$,
3. for all infinite sets of piecewise disjoint sets $A_i \in \mathfrak{A}$, $1 \leq i$, such that $\cup_{i=1}^{\infty} \in \mathfrak{A}$, μ satisfies $\mu(\cup_{i=1}^{\infty} A_i) = \sum_{i=1}^{\infty} \mu(A_i)$.

The latter property is called *countable additivity* of the measure μ.

For the purpose of our solution, we are interested in the *Lebesgue* measure. This measure is based on the length of (both open and closed) intervals. Intuitively, the length of an (open or closed) interval $[\![a, b]\!]$ is $b - a$. Before arriving at a definition for the Lebesgue measure, we introduce a number of additional concepts.

Definition 5.2.22 (*Sequential Covering Class*).
A class Λ of subsets (containing $\emptyset$) of the real line $\mathbb{R}$ is a *sequential covering class* if for every $E \subseteq \mathbb{R}$ there is a sequence $(E_n)_{n=0}^{\infty}$ of sets from Λ for which Eqn. (5.24) holds.

$$E \subseteq \cup_{n=1}^{\infty} E_n \tag{5.24}$$

Whenever Eqn. (5.24) holds for a set E and a sequence of sets $(E_n)_{n=1}^{\infty}$, we say $(E_n)_{n=1}^{\infty}$ *covers* E.

Henceforth, let the family Γ denote all countable collections of open intervals. For an arbitrary subset E of $\mathbb{R}$, we define the subfamily $\mathfrak{C}(E)$ of Γ containing the covers of E as the set $\{\gamma \in \Gamma \mid \gamma \text{ covers } E\}$.

Definition 5.2.23 (*Lebesgue Outer Measure*).
The function $\mu^*{:}2^{\mathbb{R}} \to \mathbb{R}_{\geq \mathbf{0}} \cup \{\infty\}$, defined by Eqn. (5.25), is called the *Lebesgue outer measure*.

$$\mu^*(E) = \bigsqcup \{\lambda^*(\gamma) \mid \gamma \in \mathfrak{C}(E)\} \tag{5.25}$$

where $\lambda^*(\gamma) = \sum_{I \in \gamma} \lambda(I)$ is the length of a countable collection of open intervals, $\lambda(I) = b - a$ for all intervals I with endpoints $a \leq b$ (both open and closed intervals), is the length of an interval, and $\sqcup E$ is the least upper bound of the set E.

Note that the Lebesgue outer measure μ^* extends the interval length function λ, i.e. for every interval I, we have $\mu^*(I) = \lambda(I)$. Hence, we have $\mu^*(\emptyset) = 0$. Moreover, since $\lambda(I) \geq \mathbf{0}$ for all I, we also have $\mu^*(E) \geq \mathbf{0}$ for all $E \in 2^{\mathbb{R}}$.

Definition 5.2.24 (*Lebesgue Measurable*).
A subset E of $\mathbb{R}$ is called *Lebesgue measurable* if for all sets $X \subseteq \mathbb{R}$, Eqn. (5.26) is satisfied.

$$\mu^*(X) = \mu^*(X \cap E) + \mu^*(X \setminus E) \tag{5.26}$$

Let $\mathfrak{M}$ represent the class of all measurable subsets of the real line $\mathbb{R}$. The function obtained by restricting the domain of the Lebesgue outer measure to Lebesgue measurable sets is countably additive. Since we already established the Lebesgue outer measure satisfied the first two conditions of Def. 5.2.21, the restriction defines a measure.

Definition 5.2.25 (*Lebesgue Measure*).
The function $\mu{:}\mathfrak{M} \to \mathbb{R}_{\geq \mathbf{0}} \cup \{\infty\}$, defined by Eqn. (5.27), is called the *Lebesgue measure*.

$$\mu(E) = \mu^*(E) \tag{5.27}$$

The Lebesgue measure is often used in conjunction with integration of functions that are not continuous everywhere. We employ the measure in conjunction with the differentiation of functions that are not differentiable everywhere. To be more precise, we first define what it means for a predicate to hold *almost everywhere*.

Definition 5.2.26 (*Almost Everywhere*).
Let P be a predicate on a domain X. The predicate P is said to hold *almost everywhere,* if it holds on a set $E \subseteq X$, such that $\mu(X \setminus E) = 0$. In other words, P holds if it holds everywhere except for on a set with Lebesgue measure zero.

Remark 5.2.27. *Note that Lebesgue-measurable sets with measure zero are not necessarily sets with countably many elements, e.g. the Cantor set has measure zero (this is the subset of the closed interval* $[0, 1]$ *from which the middle third is removed, repeating this process with the two remaining third pieces ad infinitum).*

Definition 5.2.28 (*Differentiable almost everywhere*).
A function f is said to be differentiable almost everywhere, denoted by $\mathsf{diff}(f)$ a.e., whenever the set of elements $A \subseteq \mathsf{dom}(f)$ for which f is not differentiable has Lebesgue measure zero.

Now recall Example 5.2.20. Even though the witness, portrayed in Fig. 5.9 is not differentiable in the strictest sense, there is only a single point ($t = 0.5$) in the (open) interval $(0, 2)$ for which this witness is not differentiable. By definition, we have $\mu(\{0.5\}) = 0$, and therefore, the witness of Fig. 5.9 is differentiable almost everywhere. In general, a sequence of piecewise differentiable functions which are not differentiable on their perimeters can be characterised by a single differentiable almost everywhere function.

Notation 5.2.29. We use the set $\mathcal{D}_d^D$, $d \geq 0$, to denote the set, consisting of all continuous functions $f{:}[0, d] \to D$ that are differentiable almost everywhere on their domain. For any function $f \in \mathcal{D}_d^D$, the set $\omega_f \subseteq (0, d)$ is the set of elements for which $\dot{f}$ exists. When, from the context, it is clear which range we use, we write $\mathcal{D}_d$ rather than $\mathcal{D}_d^D$.

In conclusion of this intermezzo, we present a semantics of a hybrid automaton in terms of a TSLTS.

Definition 5.2.30 (*Semantics of a Hybrid Automaton (TSLTS)*).
Let $X = \langle M, M^0, \Sigma, V, \iota, \theta, \mathcal{I}, E\rangle$ be a hybrid automaton. The semantics of X is given by the TSLTS $X^{\overline{s}} = \langle M \times \mathbb{V} \times \mathbb{R}_{\geq \mathbf{0}}, (M^0 \times \mathbb{V}_V \times \{\mathbf{0}\})_{\mathcal{I}}, \Sigma, \to, \mathcal{U}\rangle$. The time-stamped action transition $\to$ and the delay predicate $\mathcal{U}$ are defined as the least relations satisfying the rules of Table 5.9.

$$\frac{\mathcal{U}_u(m,\vartheta,t) \quad u' \leq u}{\mathcal{U}_{u'}(m,\vartheta,t)} \qquad \frac{f \in \mathcal{D}_d^{\mathbb{V}_V} \quad f(\mathbf{0}) = \vartheta \quad f \models \iota(m) \quad f\rceil\omega_f \models \theta(m) \quad \mathbf{0} \leq d}{\mathcal{U}_{t+d}(m,\vartheta,t)}$$

$$\frac{\begin{array}{c} f \in \mathcal{D}_d^{\mathbb{V}_V} \quad f(\mathbf{0}) = \vartheta \quad f \models \iota(m) \quad f\rceil\omega_f \models \theta(m) \\ m \xrightarrow{\sigma,\varphi,\rho} m' \qquad f(d) \models \varphi \quad \vartheta' \models \rho[f(d)] \quad \vartheta' \models \iota(m') \quad \mathbf{0} \leq d \end{array}}{(m,\vartheta,t) \xrightarrow{\sigma}_{t+d} (m',\vartheta',t+d)}$$

Table 5.9: Deduction Rules for Single-Step Evolution and Idling Behaviour

The state-space of $X^{\overline{s}}$ consists of the states that can be reached from the initial states

Definition 5.2.31 (*Timed Bisimulation for Hybrid Automata (TSLTS)*).
Let X_1 and X_2 be two hybrid automata, and let $X_1^{\overline{s}}$ and $X_2^{\overline{s}}$ be the interpretations of X_1 and X_2 in terms of Time-Stamped Labelled Transition Systems. A relation $\mathcal{R} \subseteq (M_1 \times \mathbb{V}_{V_1} \times \mathbb{R}_{\geq \mathbf{0}}) \times (M_2 \times \mathbb{V}_{V_2} \times \mathbb{R}_{\geq \mathbf{0}})$ is a timed bisimulation iff for all states (m_1, ϑ, t) and (m_2, χ, u), the following requirements are fulfilled.

1. if $(m_1, \vartheta, t)\mathcal{R}(m_2, \chi, u)$ and $(m_1, \vartheta, t) \xrightarrow{\sigma}_r (m_1', \vartheta', r)$ for some $\sigma \in \Sigma_1$ and $r \in \mathbb{R}_{\geq \mathbf{0}}$, then there exists a state (m_2', χ', u), such that $(m_2, \chi, u) \xrightarrow{\sigma}_r (m_2', \chi', r)$ and, moreover, $(m_1', \vartheta', r)\mathcal{R}(m_2', \chi', r)$. Similarly for each action transition of automaton X_2.

2. if $(m_1, \vartheta, t)\mathcal{R}(m_2, \chi, u)$ and $\mathcal{U}_r(m_1, \vartheta, t)$ for some $r \in \mathbb{R}_{\geq \mathbf{0}}$, then also $\mathcal{U}_r(m_2, \chi, u)$.

3. For each state $(m_1, \vartheta, \mathbf{0}) \in (M_1^0 \times \mathbb{V}_{V_1} \times \{\mathbf{0}\})_{\mathcal{I}_1}$, we can identify a state $(m_2, \chi, \mathbf{0}) \in (M_2^0 \times \mathbb{V}_{V_2} \times \{\mathbf{0}\})_{\mathcal{I}_2}$, such that $(m_1, \vartheta, \mathbf{0})\mathcal{R}(m_2, \chi, \mathbf{0})$, and, conversely, for each initial state of X_2 we can identify an $\mathcal{R}$-related initial state of X_1.

If there exists a relation $\mathcal{R}$, relating all states between $X_1^{\bar{s}}$ and $X_2^{\bar{s}}$ according to the above-listed criteria, then we say the two hybrid automata X_1 and X_2 are timed bisimilar, denoted by $X_1 \leftrightarroweq_t X_2$.

For all reachable and bisimilar states, two timed bisimilar hybrid automata synchronise on their elapsed time.

Lemma 5.2.32. Let X_1 and X_2 be two timed bisimilar timed automata and $\mathcal{R}$ be a relation relating X_1 and X_2 , i.e. $\mathcal{R}{:}X_1 \leftrightarroweq_t X_2$. Then, for all states (l, ϑ, t) of $X_1^{\bar{s}}$ and (s, χ, u) of $X_2^{\bar{s}}$, reachable from an initial state of X_1 and X_2, implication (5.28) holds.

$$(l, \vartheta, t)\mathcal{R}(s, \chi, u) \Rightarrow u = t \tag{5.28}$$

Proof. The proof follows the same line of reasoning held for Lemma 5.1.16. □

Lemma 5.2.33. Let X be a hybrid automaton. Then, for all states (m, ϑ, t), reachable from an initial state, we know $\mathcal{U}_u(m, \vartheta, t)$ holds for all $u \leq t$.

Proof. See the proof of Lemma 5.1.22. □

Definition 5.2.34 (*Semantics of Synchronising Hybrid Automata (TSLTS)*).
Let $X_1 = \langle M_1, M_1^0, \Sigma_1, V_1, \theta_1, \iota_1, \mathcal{I}_1, E_1\rangle$ and $X_2 = \langle M_2, M_2^0, \Sigma_2, V_2, \theta_2, \iota_2, \mathcal{I}_2, E_2\rangle$ be two hybrid automata. The semantics of $X_1 \parallel_s X_2$ is given by the TSLTS $(X_1)^{\bar{s}} \parallel_s (X_2)^{\bar{s}} = \langle (M_1 \times \mathbb{V}_{V_1} \times \mathbb{R}_{\geq \mathbf{0}}) \times (M_2 \times \mathbb{V}_{V_2} \times \mathbb{R}_{\geq \mathbf{0}}), (M_1^0 \times \mathbb{V}_{V_1} \times \{\mathbf{0}\})_{\mathcal{I}_1} \times (M_2^0 \times \mathbb{V}_{V_2} \times \{\mathbf{0}\})_{\mathcal{I}_2}, \Sigma_1 \cup \Sigma_2, \rightarrow, \mathcal{U}\rangle$. For all states $(m, \vartheta, t) \in X_1^{\bar{s}}$ and $(n, \chi, u) \in X_2^{\bar{s}}$, the set of time-stamped action transitions $\rightarrow$ and the delay predicate $\mathcal{U}$ are defined as the least relations satisfying the rules in Table 5.10.

$$\frac{\mathcal{U}_r(m, \vartheta, t) \quad \mathcal{U}_r(n, \chi, u)}{\mathcal{U}_r((m, \vartheta, t)\|_s(n, \chi, u))}$$

$$\frac{(m, \vartheta, t) \xrightarrow{\sigma}_r (m', \vartheta', r) \quad \mathcal{U}_r(n, \chi, u) \quad \sigma \in \Sigma_1 \setminus \Sigma_2 \quad u \leq r}{(m, \vartheta, t)\|_s(n, \chi, u) \xrightarrow{\sigma}_r (m', \vartheta', r)\|_s(n, \chi, u)}$$

$$\frac{(n, \chi, u) \xrightarrow{\sigma}_r (n', \chi', r) \quad \mathcal{U}_r(m, \vartheta, t) \quad \sigma \in \Sigma_2 \setminus \Sigma_1 \quad t \leq r}{(m, \vartheta, t)\|_s(n, \chi, u) \xrightarrow{\sigma}_r (m, \vartheta, t)\|_s(n', \chi', r)}$$

$$\frac{(m, \vartheta, t) \xrightarrow{\sigma}_r (m', \vartheta', r) \quad (n, \chi, u) \xrightarrow{\sigma}_r (n', \chi', r) \quad \sigma \in \Sigma_1 \cap \Sigma_2}{(m, \vartheta, t)\|_s(n, \chi, u) \xrightarrow{\sigma}_r (m', \vartheta', r)\|_s(n', \chi', r)}$$

Table 5.10: Rules for the Synchronisation Operator

Lemma 5.2.35. Let X_1 and X_2 be two hybrid automata, and let $X_1^{\bar{s}}$ and $X_2^{\bar{s}}$ be the TSLTS associated to X_1 and X_2, then there exists a timed bisimulation relation $\mathcal{R}$, such that (5.29) holds.

$$\mathcal{R}{:}X_1^{\bar{s}}\|_s X_2^{\bar{s}} \leftrightarroweq_t (X_1\|_s X_2)^{\bar{s}} \tag{5.29}$$

Proof. The proof is not too difficult (and not much fun either) and follows the exact same lines of the proof of Lemma 5.1.27. The witnessing timed bisimulation relation is defined as $\mathcal{R} = \mathcal{R}' \cap ((R_{X_1} \times R_{X_2}) \times R_{X_1 \|_s X_2})$, where $\mathcal{R}'$ is defined in Eqn. (5.30), and R_Y represents the set of reachable states of automaton Y.

$$\begin{array}{rcl} \mathcal{R} & = & \{\langle (m,\vartheta,t)\|_s(n,\chi,u)\ ,\ ((m,n),\vartheta \cup \chi',t)\rangle \mid u \leq t \wedge \\ & & \quad \exists_{f \in \mathcal{D}_{t-u}} (f\rceil\omega_f \models \theta_2(n) \wedge f \models \iota_2(n) \wedge f(\mathbf{0}) = \chi \wedge f(t-u) = \chi')\} \\ & \cup & \{\langle (m,\vartheta,t)\|_s(n,\chi,u)\ ,\ ((m,n),\vartheta' \cup \chi,u)\rangle \mid t \leq u \wedge \\ & & \quad \exists_{f \in \mathcal{D}_{u-t}} (f\rceil\omega_f \models \theta_1(m) \wedge f \models \iota_1(m) \wedge f(\mathbf{0}) = \vartheta \wedge f(u-t) = \vartheta')\} \end{array} \tag{5.30}$$

□

5.2.4 Semantics — Unity and Discord

In this section, we set out to investigate the correspondences between the standard semantics and the alternative semantics of hybrid automata. More importantly, we identify a discomforting dissimilarity between the standard semantics and the alternative semantics. An investigation of this problem automatically leads to a classification of the family of hybrid automata for which the two types of semantics agree.

Recall Example 5.2.20. This thermostat example was used to show that the flow condition associated to a control mode of a Hybrid Automaton can specify piecewise differentiable witnesses that, combined together as a single function do not yield a new differentiable function. Therefore, requiring the existence of at least a single differentiable function when idling is a too severe restriction. In our alternative semantics, this is solved by allowing witnesses that are differentiable almost everywhere. Unfortunately, there is a catch to this solution; this is illustrated below.

Example 5.2.36. Let X be the hybrid automaton depicted in Fig. 5.8, modelling the thermostat and let Y be the hybrid automaton, depicted in Fig. 5.11. Automaton Y represents a thermostat that is a variation on the thermostat modelled by automaton X, in which the draught in the room is assumed to have a gradual, rather than sudden effect on the temperature in the room. The effect of the draught is experienced as a continuous change, bounded by the flow conditions for draughty and non-draughty in automaton X. From the perspective of the standard semantics, automaton X and Y are different. This is illustrated nicely by Fig. 5.12: the witness drawn in a continuous line in Fig. 5.12 represents a course of temperature, only possible in automaton Y, whereas the witness, drawn in a dotted line in Fig. 5.12 represents a course of temperature possible in both automata X and Y. In fact, this latter witness is composed of two witnesses, i.e. during the interval $[0, 0.2]$ there is apparently no draught in the room, and in the interval $[0.2, 1]$ the draught is present. It must be noted, though, that each witness of automaton Y can be approximated by a sequence of witnesses of automaton X, and therefore, the set of reachable states are equivalent. We formalise this at the end of this chapter.

In short, automaton X and Y do not agree on each others delay transitions in the standard semantics. Quite the reverse can be said for the alternative semantics, which readily equates automata X and Y. In short, this is because the set of states, reachable by differentiable almost everywhere functions for automaton X is the same as the set of states, reachable by differentiable almost everywhere functions for automaton Y.

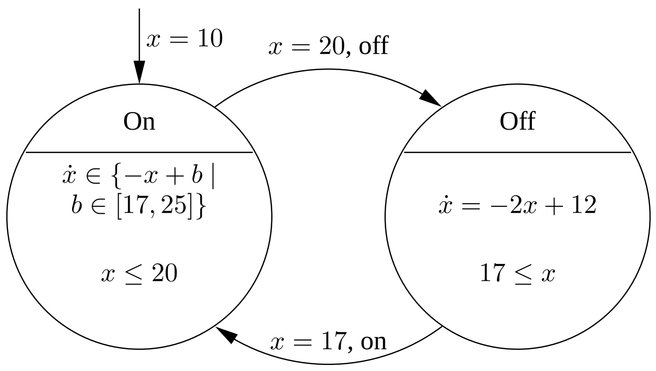


Figure 5.11: Thermostat revisited

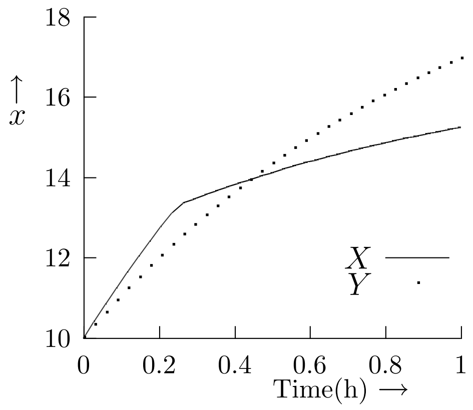


Figure 5.12: Witnesses for X and Y

From the effect described by the previous example, it follows that hybrid automata, interpreted in their standard semantics, do not satisfy *time additivity*, i.e. it is not guaranteed that by successively delaying a total of d time units, the system could have reached the same state as when delaying d time units at once. Note that the reverse, viz. *time-interpolation* does hold for the standard semantics: when delaying d time units at once, we end up in a state that could also have been reached by successively delaying a total of d time units. Finally, both the standard and the alternative semantics do not necessarily guarantee time-determinism.

In conclusion, the results from the above example and discussion are summarised by Theorem 5.2.37.

Theorem 5.2.37 The TPLTS semantics of a hybrid automaton does in general not guarantee time-additivity; the TSLTS semantics of a hybrid automaton necessarily guarantees time-additivity.

Remark 5.2.38. *The statement of Theorem 5.2.37 explicitly does not claim that time-additivity never is guaranteed by the TPLTS semantics of a hybrid automaton. For instance, timed automata, which can be (bi)simulated by hybrid automata, do satisfy time-additivity.*

The remainder of this section is devoted to relating the two semantics. Notice that there is no possibility of adapting the TSLTS semantics of a hybrid automaton to coincide with the TPLTS semantics. Hence, we are left to pursue the following options:

1. Coarsen the timed bisimulation relation for the TPLTS semantics of a hybrid automaton,
2. Classify a family of hybrid automata for which both the TPLTS semantics and the TSLTS semantics coincide.

Additive Timed Bisimilarity

Following [125], we introduce a new bisimulation relation that is coarser than timed bisimulation for the TPLTS semantics. The new bisimulation, hereafter referred to as the *additive timed bisimulation*, considers the reflexive-transitive closure of delay transitions of a TPLTS. Using additive timed bisimulation ensures time-additivity is the only obstacle preventing the identification of the TPLTS semantics with the TSLTS semantics.

Notation 5.2.39. We use $\overset{t}{\longmapsto}^*$ to denote a sequence of delay transitions, the last of which occurs at (absolute) point of time t.

Definition 5.2.40 (*Additive Timed Bisimulation for Hybrid Automata*).
Let X_1 and X_2 be two hybrid automata, and let X_1^s and X_2^s be the interpretations of X_1 and X_2 in terms of TPLTSs. A relation $\mathcal{R} \subseteq (M_1 \times \mathbb{V}_{V_1} \times \mathbb{R}_{\geq \mathbf{0}}) \times (M_2 \times \mathbb{V}_{V_2} \times \mathbb{R}_{\geq \mathbf{0}})$ is a additive timed bisimulation iff for all states (m_1, ϑ, t) and (m_2, χ, u), the following requirements are satisfied.

1. if $(m_1, \vartheta, t)\mathcal{R}(m_2, \chi, u)$ and $(m_1, \vartheta, t) \xrightarrow{\sigma} (m_1', \vartheta', t)$ for some $\sigma \in \Sigma_1$, then there exists a state (m_2', χ', u), such that $(m_2, \chi, u) \xrightarrow{\sigma} (m_2', \chi', u)$ and $(m_1', \vartheta', t)\mathcal{R}(m_2', \chi', u)$. Similar for each action transition of automaton X_2.

2. if $(m_1, \vartheta, t)\mathcal{R}(m_2, \chi, u)$ and $(m_1, \vartheta, t) \overset{t+d}{\longmapsto} (m_1, \vartheta', t + d)$ for some $d \in \mathbb{R}_{\geq \mathbf{0}}$, then there exists a state $(m_2, \chi', u + d)$, such that $(m_2, \chi, u) \overset{u+d}{\longmapsto}^* (m_2, \chi', u + d)$ and, moreover, $(m_1, \vartheta', t + d)\mathcal{R}(m_2, \chi', u + d)$. Similarly for each delay transition of automaton X_2.

3. For each state $(m_1, \vartheta, \mathbf{0}) \in (M_1 \times \mathbb{V}_{V_1} \times \{\mathbf{0}\})_{\mathcal{I}_1}$, we can identify a related initial state $(m_2, \chi, \mathbf{0}) \in (M_2 \times \mathbb{V}_{V_2} \times \{\mathbf{0}\})_{\mathcal{I}_2}$ such that $(m_1, \vartheta, \mathbf{0})\mathcal{R}(m_2, \chi, \mathbf{0})$. Likewise for each initial state of X_2.

Whenever there exists a relation $\mathcal{R}$, satisfying the above requirements, we say the two hybrid automata X_1 and X_2 are additive timed bisimilar, denoted by $\mathcal{R}{:}X_1 \underline{\leftrightarrow}_t^* X_2$.

Corollary 5.2.41 For all hybrid automata X_1 and X_2, yielding TPLTSs satisfying time additivity, and all relations $\mathcal{R}$, Eqn. (5.31) holds.

$$\mathcal{R}{:}X_1 \underline{\leftrightarrow}_t^{\iota} X_2 \;\Leftrightarrow\; \mathcal{R}{:}X_1 \underline{\leftrightarrow}_t^* X_2 \tag{5.31}$$

Proof. Follows immediately from the definition of time additivity and additive timed bisimulation. □

The standard semantics and the alternative semantics are related via the following two lemmata.

Lemma 5.2.42. For all hybrid automata X and delays d, Eqn. (5.32) holds.

$$\mathcal{U}_{t+d}(m, \vartheta, t) \;\Leftrightarrow\; \exists_{\vartheta'}(m, \vartheta, t) \overset{t+d}{\longmapsto}^* (m, \vartheta', t + d) \tag{5.32}$$

Proof. Suppose $\mathcal{U}_{t+d}(m, \vartheta, t)$ holds. This means we can find a function $f \in \mathcal{D}_d$, such that $f(\mathbf{0}) = \vartheta$, $f \models \iota(m)$ and $f\rceil\omega_f \models \theta(m)$. Let $\overline{\omega}_f = [\mathbf{0}, d] \setminus \omega_f$. Note, that by definition of ω_f, we have $\{\mathbf{0}, d\} \subseteq \overline{\omega}_f$. Let $[\overline{\omega}_f]_n$ denote the n^{th} element of $\overline{\omega}_f$ when ordered according to $\leq$, i.e. $[\overline{\omega}_f]_1 = \mathbf{0}$. Define the functions $F_n{:}[[\overline{\omega}_f]_n, [\overline{\omega}_f]_{n+1}] \to \mathbb{V}_V$ for all $n < |\overline{\omega}_f|$, as $F_n(\varepsilon) = f(\varepsilon)$. By construction, we now have $(F_n)_{n=1}^{|\overline{\omega}_f - 1|} = f$. We know f is continuous on $[0, d]$; therefore, also all functions F_n are continuous on their domain. Moreover, f is differentiable almost everywhere. More concretely, we know f is differentiable on ω_f. This means that f is strictly differentiable on the open interval $([\overline{\omega}_f]_n, [\overline{\omega}_f]_{n+1})$ for all $n < |\overline{\omega}_f|$. Hence, F_n is differentiable on its domain, except for the begin and end-points of its interval. Lastly, since $f \models \iota(m)$, we also know $F_n \models \iota(m)$ for all $n < |\overline{\omega}_f|$. Summarising, we know there exists a delay transition delaying to time $t_{n+1} = t + [\overline{\omega}_f]_{n+1}$, i.e. $(m, \vartheta, t) \overset{t_{n+1}}{\longmapsto} (m, F_n[\overline{\omega}_f]_{n+1}, t + [\overline{\omega}_f]_{n+1})$ for all $n < |\overline{\omega}_f|$. Note that $t + d = t_{[\overline{\omega}_f]_{|\overline{\omega}_f|}}$, which proves the implication of (5.32).

Conversely, assume $\exists_{\vartheta'}(m, \vartheta, t) \overset{t+d}{\longmapsto}{}^{*} (m, \vartheta', t + d)$. For each transition, we know there exists a witness. Let $f_n{:}[\mathbf{0}, d_n] \to \mathbb{V}_V$ denote the witness of the n^{th} delay transition in the above sequence and let f_N be the witness for the last delay transition in this sequence. We then know that $\mathsf{diff}(f_n)$, $\mathsf{cont}(f_n)$, $f_1(\mathbf{0}) = \vartheta$, $f_{n+1}(\mathbf{0}) = f_n(d_n)$, $f_N(d) = \vartheta'$, $f_n \models \iota(m)$ and $f_n\rceil(\mathbf{0}, d_n) \models \theta(m)$. Define $g_{n+1}{:}[\sum_{i=1}^{n} d_i, \sum_{i=1}^{n+1} d_i] \to \mathbb{V}_V$ as $g_{n+1}((\sum_{i=1}^{n} d_i) + \varepsilon) = f_{n+1}(\varepsilon)$ for all $0 \leq n < N$, i.e. $g_1 = f_1$ and $g_n(\sum_{i=1}^{n} d_i) = g_{n+1}(\sum_{i=1}^{n} d_i)$. Define the function $F{:}[0, d] \to \mathbb{V}_V$ as $F = (g_n)_{n=1}^{N}$. By construction, F is continuous. Now suppose F is not differentiable almost everywhere. That means there is an interval $[\![b, e]\!] \subseteq [0, d]$, for which F is not differentiable, and $\mu([\![b, e]\!]) > 0$. This means that we can identify a function g_n such that $\mu([\![b, e]\!] \cap \mathsf{dom}(g_n)) > 0$ that is not differentiable. But since all g_n are differentiable on their open domains, this cannot be the case. Hence, such an interval does not exist and therefore, F is differentiable almost everywhere. Moreover, $F(\mathbf{0}) = \vartheta$ and $F(d) = \vartheta'$, $F \models \iota(m)$ and $F\rceil\omega_F \models \theta(m)$. Hence, also $\mathcal{U}_{t+d}(m, \vartheta, t)$.
□

Lemma 5.2.43. For all hybrid automata X and delays d, Eqn. (5.33) holds.

$$(m, \vartheta, t) \xrightarrow{\sigma}_{t+d} (m', \vartheta', t{+}d) \Leftrightarrow \exists_{\vartheta''}(m, \vartheta, t) \overset{t+d}{\longmapsto}{}^{*} (m, \vartheta'', t{+}d) \xrightarrow{\sigma} (m', \vartheta', t{+}d) \quad (5.33)$$

Proof. Suppose $(m, \vartheta, t) \xrightarrow{\sigma}_{t+d} (m', \vartheta', t + d)$. Let $f{:}[\mathbf{0}, d] \to \mathbb{V}_V$ be the witness for the time-stamped action transition, i.e. we have $f \models \iota(m)$. Then, we know there exists a switch $m \xrightarrow{\sigma,\varphi,\rho} m'$, such that $\mathcal{U}_{t+d}(m, \vartheta, t)$, $f(d) \models \varphi$, $\vartheta' \models \rho[f(d)]$ and $\vartheta' \models \iota(m')$. Moreover, if we re-examine the proof of Lemma 5.2.42, we know in fact that $(m, \vartheta, t) \overset{t+d}{\longmapsto}{}^{*} (m, f(d), t + d)$. But then we can also derive $(m, f(d), t + d) \xrightarrow{\sigma} (m', \vartheta', t + d)$.

Conversely, assume there exists a valuation ϑ'', such that $(m, \vartheta, t) \overset{t+d}{\longmapsto}{}^{*} (m, \vartheta'', t + d) \xrightarrow{\sigma} (m', \vartheta', t + d)$. Then we know there is a switch $m \xrightarrow{\sigma,\varphi,\rho} m'$, such that $\vartheta'' \models \varphi$, $\vartheta' \models \rho[\vartheta'']$ and $\vartheta' \models \iota(m')$. From Lemma 5.2.42, we also know $\mathcal{U}_{t+d}(m, \vartheta, t)$ holds, and in fact, there is a witness $F{:}[\mathbf{0}, d] \to \mathbb{V}_V$, such that $F(d) = \vartheta''$. Hence, we also know a time-stamped action transition $(m, \vartheta, t) \xrightarrow{\sigma}_{t+d} (m', \vartheta', t + d)$ is possible. □

Theorem 5.2.44 For all hybrid automata X_1 and X_2, Eqn. (5.34) holds.

$$X_1 \leftrightarroweq{}_t^{*} X_2 \;\Leftrightarrow\; X_1 \leftrightarroweq{}_t X_2 \quad (5.34)$$

Proof. Suppose $\mathcal{R}{:}X_1 \leftrightarroweq{}_t^{*} X_2$; using Lemmata 5.2.42 and 5.2.43, the proof proceeds according to the lines of the proof for Theorem 5.1.30 showing that also $\mathcal{R}{:}X_1 \leftrightarroweq{}_t X_2$.
Conversely, let $\mathcal{R}^*$ be the union of all timed bisimulation relations $\mathcal{R}_i{:}X_1 \leftrightarroweq{}_t X_2$. Then, also $\mathcal{R}^*{:}X_1 \leftrightarroweq{}_t X_2$. Moreover, also $\mathcal{R}^*{:}X_1 \leftrightarroweq{}_t^{*} X_2$. For the initial states, this statement holds by definition. Hence, it suffices to examine the delay and action transitions.

1. Let $(m, \vartheta, t)\mathcal{R}^*(n, \chi, t)$ and suppose $(m, \vartheta, t) \overset{t+d}{\longmapsto} (m, \vartheta', t{+}d)$. Using Lemma 5.2.42, we know $\mathcal{U}_{t+d}(m, \vartheta, t)$ and, since $\mathcal{R}^*$ is a timed bisimulation relation, also $\mathcal{U}_{t+d}(n, \chi, t)$. Since

state $(m, \vartheta', t+d)$ is reachable, we know $(m, \vartheta', t+d) \overset{t+d}{\longmapsto} (m, \vartheta', t+d)$ and, therefore $\mathcal{U}_{t+d}(m, \vartheta', t+d)$. This means there exists a state $(n, \chi', t+d)$, such that $\mathcal{U}_{t+d}(n, \chi', t+d)$ and $(m, \vartheta', t+d)\mathcal{R}^*(n, \chi', t+d)$. By Lemma 5.2.42, we know there exists a state $(n, \chi'', t+d)$, such that $(n, \chi, t) \overset{t+d}{\longmapsto}^* (n, \chi'', t+d)$. Now, either also $(m, \vartheta', t+d)\mathcal{R}^*(n, \chi'', t+d)$, and we are done, or $(m, \vartheta', t+d) \not\mathcal{R}^*(n, \chi'', t+d)$. But this latter option is not possible, since that would mean that state (n, χ, t) has other characteristics than (m, ϑ, t) and is never related via any of the timed bisimulation relations $\mathcal{R}_i$. Thus, also $(m, \vartheta', t+d)\mathcal{R}^*(n, \chi'', t+d)$. Similarly for a timed transition for X_2.

2. Let $(m, \vartheta, t)\mathcal{R}^*(n, \chi, t)$ and suppose $(m, \vartheta, t) \xrightarrow{\sigma} (m', \vartheta', t)$. Then, also $(m, \vartheta, t) \overset{t}{\longmapsto} (m, \vartheta, t)$ and thus by Lemma 5.2.43 also $(m, \vartheta, t) \xrightarrow{\sigma}_t (m', \vartheta', t)$. But then we also have $(n, \chi, t) \xrightarrow{\sigma}_t (n', \chi', t)$ for some (n', χ', t) such that $(m', \vartheta', t)\mathcal{R}^*(n', \chi', t)$. By Lemma 5.2.43, we then also have $(n, \chi, t) \xrightarrow{\sigma} (n', \chi', t)$.

□

We thus have proved that timed bisimilarity modulo time additivity for the TPLTS semantics of a hybrid automaton coincides with timed bisimilarity for the TSLTS semantics of a hybrid automaton. This means that, apart from the time additivity there is no real semantical difference between the standard semantics and the alternative semantics.

Approximable Convex Flow Constraints

The previous section established that time additivity is the only property that prevents us from identifying the standard semantics with the alternative semantics. As we have already seen, there is a class of hybrid automata for which their TPLTSs do guarantee time additivity (e.g. hybrid automata with constant flows, such as timed automata). In this section, we narrow down the conditions under which the TPLTS semantics of a hybrid automaton is necessarily time additive.

The problem with the time additivity property is rooted in the flow constraint of a control mode in a hybrid automaton: this constraint determines the shape and characteristics of the witnesses. Problems only arise when the composition of two witnesses can reach valuations a single witness can never reach, as illustrated by Example 5.2.36. This is formalised in Lemma 5.2.46.

Definition 5.2.45 (*Approximability of Flow Constraints*).
Let θ be a flow constraint. Then, θ is approximable, iff for all functions $f \in \mathcal{D}_d^{\mathbb{V}_V}$, such that $f\rceil(0,d) \models \theta$, a function $F{:}[\mathbf{0}, d] \to \mathbb{V}_V$ with $F(\mathbf{0}) = f(\mathbf{0})$ and $F(d) = f(d)$ exists satisfying $\mathsf{cont}(F)$, $\mathsf{diff}(F\rceil(0,d))$ and $F\rceil(0,d) \models \theta$.

Lemma 5.2.46. Let X be a hybrid automaton. Then the TPLTS of X guarantees time additivity iff all for all $m \in M$ $\theta(m)$ is approximable.

Proof. Let X be a hybrid automaton and let m be a control mode of X. Suppose the TPLTS of X guarantees time additivity. Let $d \geq \mathbf{0}$, $f \in \mathcal{D}_d$ and $\vartheta \in \mathbb{V}_V$, such that $f(\mathbf{0}) = \vartheta$, $f \models \iota(m)$ and $f\rceil(0,d) \models \theta(m)$. The deduction rules for the alternative semantics then give $\mathcal{U}_{t+d}(m, \vartheta, t)$, and using Lemma 5.2.42, we also know there is a sequence of delay transitions that leads to the state $(m, f(d), t+d)$. In other words, we have $(m, \vartheta, t) \overset{t+d}{\longmapsto}^* (m, f(d), t+d)$.

Since the transition system guarantees time additivity, we therefore also know there is a single transition $(m, \vartheta, t) \overset{t+d}{\longmapsto} (m, f(d), t + d)$. This, however, gives rise to the existence of a function F, satisfying $F\mathbf{0} = \vartheta$, $\mathsf{cont}(F)$, $\mathsf{diff}(F\rceil(\mathbf{0}, d))$, $F \models \iota(m)$ and $F\rceil(\mathbf{0}, d) \models \theta(m)$.
Conversely, suppose that for all $d \geq \mathbf{0}$, $f \in \mathcal{D}_d$ and $\vartheta \in \mathbb{V}_V$, such that $f\mathbf{0} = \vartheta$, $f \models \iota(m)$ and $f\rceil(\mathbf{0}, d) \models \theta(m)$, there is a function $F{:}[\mathbf{0}, d] \to \mathbb{V}_V$ satisfying $F(\mathbf{0}) = \vartheta$, $\mathsf{cont}(F)$, $\mathsf{diff}(F\rceil(\mathbf{0}, d))$, $F \models \iota(m)$, $F\rceil(\mathbf{0}, d) \models \theta(m)$ and $F(d) = f(d)$. Let $(m, \vartheta, t) \overset{t+d}{\longmapsto}{}^{*} (m, \vartheta', t+d)$. Using Lemma 5.2.42, we then know $\mathcal{U}_{t+d}(m, \vartheta, t)$ holds. But that means there is a witness $g \in \mathcal{D}_d$, such that $g(\mathbf{0}) = \vartheta$, $g \models \iota(m)$ and $g\rceil(\mathbf{0}, d) \models \theta(m)$. Moreover, following the construction of the witness as is done in the proof of Lemma 5.2.42, we know that $g(d) = \vartheta'$. Now, according to our assumptions, this means there is a function $F{:}[\mathbf{0}, d] \to \mathbb{V}_V$, witnessing the transition $(m, \vartheta, t) \overset{t+d}{\longmapsto} (m, \vartheta', t + d)$. But this is exactly what is necessary for time additivity.
□

Definition 5.2.47 (*Approximable Hybrid Automata*).
The hybrid automaton X is approximable iff all reachable control modes m have flow conditions $\theta(m)$ that are approximable.

Examples of approximable hybrid automata are e.g. *linear hybrid automata* [9] and *rectangular automata* [63], but also of course, timed automata [8]. Finding more general sufficient and necessary conditions that allow us to derive, based on the appearance of the flow conditions, whether a hybrid automaton is an approximable hybrid automaton is still an open problem.

Theorem 5.2.48 For all approximable hybrid automata X_1 and X_2, Eqn (5.35) holds.

$$X_1 \underline{\leftrightarrow}_t^{*} X_2 \;\Leftrightarrow\; X_1 \underline{\leftrightarrow}_t^{\iota} X_2 \;\Leftrightarrow\; X_1 \underline{\leftrightarrow}_t X_2 \tag{5.35}$$

Proof. Follows immediately from Theorem 5.2.44 and Corollary 5.2.41. □

One may argue that approximable hybrid automata might not occur in practice, and thence, Theorem 5.2.48 is in some sense void. However, all acceptable hybrid automata, such as linear hybrid automata and rectangular automata fall into the category of approximable hybrid automata. These automata are frequently used in practice; this means that approximable hybrid automata encompass not just another theoretical classification of systems, but can actually be used in practice.

Chapter 6

Interpretations of Automata

In this chapter, we investigate the relation between timed automata and hybrid automata on the one hand and μCRL$_t$ on the other hand. The results of this investigation provide an insight in the expressivity of timed automata, hybrid automata and μCRL$_t$.

The relation between timed automata and other formalisms has received prior attention. Fokkink [43], for instance, suggests a translation from strongly regular ACPrI processes to timed automata and vice versa. In [44], Fokkink provides an outline for an interpretation of timed automata into ACP with prefix integration and shows the latter is strictly more expressive. The timed automata he considers, however, are the original timed automata proposed by Alur and Dill [7, 8]. These automata do not allow for the execution of two successive actions at the same point in time, nor do they incorporate invariants to restrict the time that can be spent in a location. As such, the relations established in [43] and [44] do not carry over to our setting.

D'Argenio [38], studies timed automata by describing a dedicated process algebra for a variant of timed (safety) automata [65]. The semantics of this process algebra is defined in terms of timed automata. Soundness questions are addressed, but no completeness. A similar approach is undertaken by [83], where a sound and complete axiomatisation of timed automata with invariants is described.

As far as we can tell, no prior attempt has been made in comparing hybrid automata to other formalisms. Since we are only slowly progressing towards a deeper understanding of hybrid systems, such investigations are important, both from a theoretical and practical point of view. For instance, time as an entity, plays a pivotal rôle in both real-time discrete systems and purely continuous systems. It is, however, not entirely clear whether the properties we appertain to time in real-time discrete systems match those that are understood for purely continuous systems.

Apart from the insight we can gain into the formalisms, the practical aspects are that any relation between timed automata and μCRL$_t$ has implications for the development of tool support for μCRL$_t$. For instance, the relation we establish in this chapter suggests that, for a class of (timed) μCRL$_t$ expressions, model checking is viable, even for a class of μCRL$_t$ expressions representing hybrid systems.

This chapter is divided in two parts. We first extensively discuss the interpretations of timed automata in μCRL$_t$ and the soundness of these interpretations, and then repeat the exercise for hybrid automata in slightly less detail.

6.1 Interpreting Timed Automata

Section 6.1.1 introduces a schema for translating timed automata and synchronising timed automata to μCRL$_t$. The correctness of this translation is investigated in Section 6.1.2. Section 6.1.3 then summarises the results and implications for the preceding sections.

6.1.1 Interpretation Functions

A timed automaton is often visualised as a directed graph, where the nodes and edges represent the locations and the switches. This representation is usually preferred over the textual representation, as the graphical representation is closer to human intuition. Intuition, however, is easily misled, as Example 6.1.1 demonstrates. Observe that, in this particular example, the reason for the misconception is likely to be due to the semantics of timed automata, rather than the graphical representation.

Example 6.1.1. Consider the following process as part of a larger specification of a real-time safety critical system. On start-up, the subsystem can start executing a task within 5 time-units, and, upon completion, return to the initial state. This routine can be overridden by another subsystem, using an interrupt, after which the subsystem can return control to our subsystem under investigation. However, if the control to the subsystem is not returned within 50 time-units (measured from the start of the system), the subsystem hangs. The unaware reader might, on the basis of this explanation, agree Fig. 6.1 captures the essence of this subsystem. However, the

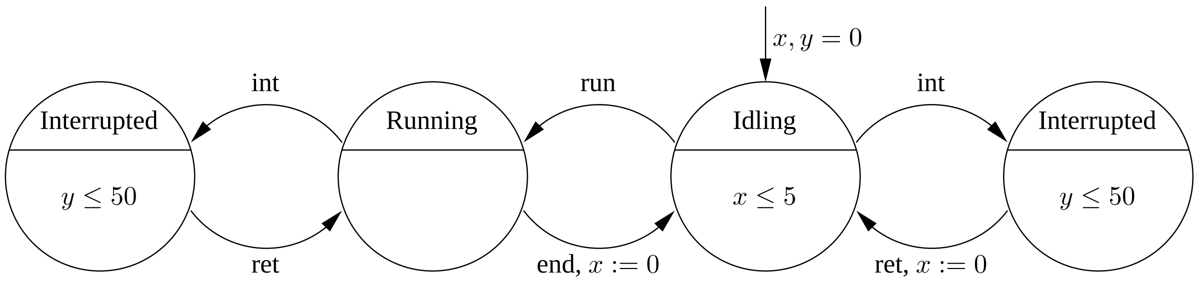


Figure 6.1: Incorrect Model for Real-Time Subsystem X

semantics associated to a timed automaton disagrees with this modelling. In fact, on the basis of this semantics, we can derive the temporal formula $\Box(\mathit{int} \Rightarrow \Diamond \mathit{ret})$, since the interrupt cannot occur at any (global) time greater than 50. Hence, the model of Fig. 6.2 is the right model for our subsystem.

Errors in models, such as the one in Example 6.1.1 may be rather uncommon (although no data is available to either refute or support this); this does not, however, imply the semantics of a timed automaton is clear.

Before we define a translation from timed automata to μCRL$_t$, we introduce shorthand-notations for frequently used entities.

Notation 6.1.2. For a timed automaton $X = \langle L, L^0, \Sigma, C, \iota, E\rangle$, we write E_l for the set of switches originating from location l. If e is a switch in E_l, then by φ_e we mean the guard of e, by σ_e we mean the action associated to switch e, by λ_e we mean the resets of e and by l_e, we mean the target location of e.

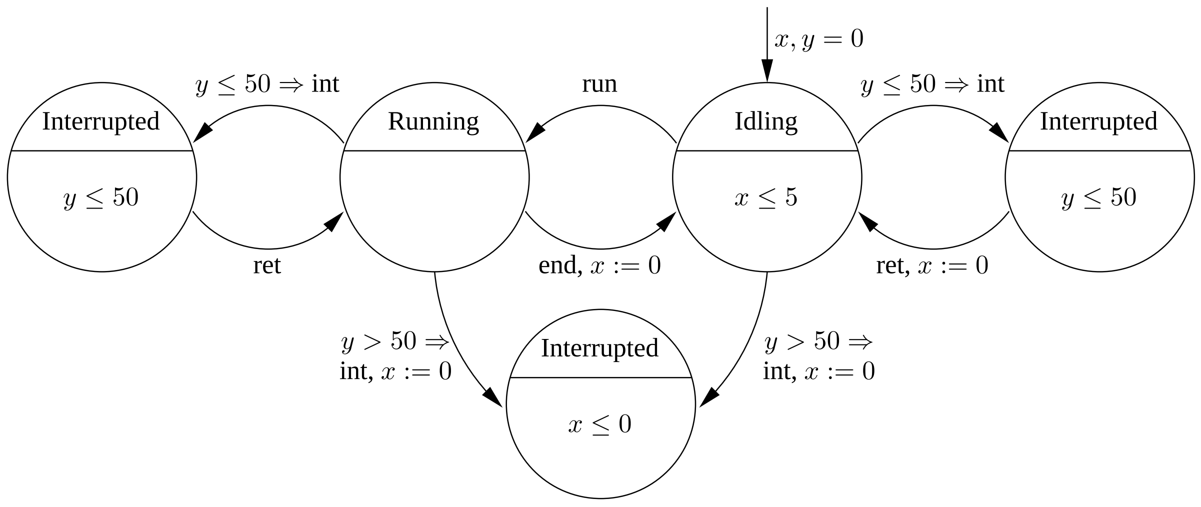


Figure 6.2: Correct Model for Real-Time Subsystem X

The translation of a timed automaton to a μCRL$_t$ process is fairly straightforward, and amenable for automation. This would allow one to analyse a timed automaton using μCRL tools.

Definition 6.1.3 (*μCRL$_t$-Interpretation of a Timed Automaton, Location Interpretation*).
Let $X = \langle L, L^0, \Sigma, C, \iota, E\rangle$ be a timed automaton. Define $[\![X]\!]{:}\mathbb{P}$ as $[\![X]\!] = \sum_{l \in L^0} X_l(\mathbf{0}, \vartheta_{\mathbf{0}})$, where $\vartheta_{\mathbf{0}} \in \mathbb{V}_C^{\mathbf{0}}$ and X_l for all $l \in L$ is defined by Eqn. (6.1).

$$\begin{aligned} X_l \;=\; & \lambda t{:}\mathbb{R}_{\geq \mathbf{0}}.\lambda\vartheta{:}\mathbb{V}_C.\ (t \gg \textstyle\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} [\vartheta, (\vartheta + d) \models \iota(l)] :: \!\longrightarrow \\ & \textstyle\sum_{e \in E_l} (\sigma_e \cdot X_{l_e}(t + d, (\vartheta + d)[\lambda_e := \mathbf{0}]) \\ & \quad \lhd \vartheta + d \models \varphi_e \wedge (\vartheta + d)[\lambda_e := \mathbf{0}] \models \iota(l_e) \rhd \delta\ ){}^{\varsigma}(t + d)) \end{aligned} \tag{6.1}$$

The process $[\![X]\!]$ is called the μCRL$_t$-interpretation of automaton X. For a timed automaton X and a location l of this automaton X, the process X_l is called the *location interpretation*.

As an example of a translation, we use the timed automaton model of Example 6.1.1 as a source.

Example 6.1.4. Let X be the timed automaton depicted in Fig. 6.1. Its μCRL$_t$-interpretation is provided in Eqn. (6.2).

$$\begin{aligned} [\![X]\!] \quad &= X_{\mathit{Idling}}(\mathbf{0}, \mathbf{0}) \\ X_{\mathit{Idling}}(x, y{:}\mathbb{R}_{\geq \mathbf{0}}) \quad &= y \gg \textstyle\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} [x, x + d \leq 5] :: \!\longrightarrow \\ &\quad (\mathit{run} \cdot X_{\mathit{Running}}(x + d, y + d) + \\ &\quad \mathit{int} \cdot X_{\mathit{Interrupted}_i}(x + d, y + d) \lhd y + d \leq 50 \rhd \delta){}^{\varsigma}(y + d) \\ X_{\mathit{Running}}(x, y{:}\mathbb{R}_{\geq \mathbf{0}}) \quad &= y \gg \textstyle\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \\ &\quad (\mathit{end} \cdot X_{\mathit{Idling}}(\mathbf{0}, y + d) + \\ &\quad \mathit{int} \cdot X_{\mathit{Interrupted}_r}(x + d, y + d) \lhd y + d \leq 50 \rhd \delta){}^{\varsigma}(y + d) \\ X_{\mathit{Interrupted}_i}(x, y{:}\mathbb{R}_{\geq \mathbf{0}}) \quad &= y \gg \textstyle\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} [y + d \leq 50] :: \!\longrightarrow \\ &\quad \mathit{ret}{}^{\varsigma}(y + d) \cdot X_{\mathit{Idling}}(\mathbf{0}, y + d) \\ X_{\mathit{Interrupted}_r}(x, y{:}\mathbb{R}_{\geq \mathbf{0}}) \quad &= y \gg \textstyle\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} [y + d \leq 50] :: \!\longrightarrow \\ &\quad \mathit{ret}{}^{\varsigma}(y + d) \cdot X_{\mathit{Running}}(x + d, y + d) \end{aligned} \tag{6.2}$$

The above depicted μCRL$_t$-interpretation immediately reveals the modelling error: action *int* is (in all location-interpretations) guarded by the condition $y + d \leq 50$. This effectively disallows any occurrences of action *int* after (in this case) absolute point in time 50.

Timed Automata can be composed using the operator $\|_s$ only. The resulting synchronising timed automaton can again be written as a timed automaton. This means that we have an indirect means of interpreting synchronising timed automata. A direct translation of a synchronising timed automaton, can also be defined. Of course, the two different translations should yield the same result. Soundness of these translations is discussed in Section 6.1.2.

Definition 6.1.5 (*μCRL$_t$-Interpretation of a Synchronising Timed Automaton*).
Let $X_1 = \langle L_1, L_1^0, \Sigma_1, C_1, \iota_1, E_1\rangle$ and $X_2 = \langle L_2, L_2^0, \Sigma_2, C_2, \iota_2, E_2\rangle$ be timed automata. The μCRL$_t$-interpretation of the synchronising timed automaton $X_1\|_s X_2$ is defined by Eqn. (6.3).

$$[\![X_1\|_s X_2]\!] = \rho_R(\ \partial_{\Sigma_1 \cap \Sigma_2}\ ([\![X_1]\!] \parallel [\![X_2]\!])) \tag{6.3}$$

Here, we define the set $\overline{\Sigma}_1 \cap \overline{\Sigma}_2 = \{\overline{\sigma} \mid \sigma \in \Sigma_1 \cap \Sigma_2\}$ and we define the function $R{:}\overline{\Sigma}_1 \cap \overline{\Sigma}_2 \to (\Sigma_1 \cap \Sigma_2)$ as $R(\overline{\sigma}) = \sigma$. Communication is defined by the function $\gamma(\sigma,\sigma) = \overline{\sigma}$ and δ otherwise.

We illustrate the translation by extending Example 6.1.1.

Example 6.1.6. Suppose a subsystem, interrupting the subsystem of Example 6.1.1, executes a single task for which it needs exactly 5 time-units. After these 5 time-units, control is again returned to the subsystem. A timed automaton Y, modelling this subsystem, is given in Fig. 6.3.

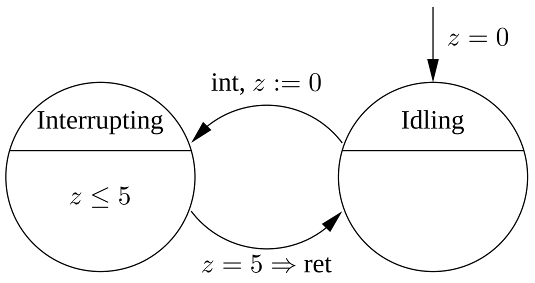


Figure 6.3: Interrupt

The μCRL$_t$-interpretation of the timed automaton Y is given by Eqn. (6.4).

$$\begin{array}{lcl} [\![Y]\!] & = & Y_{\mathit{Idling}}(\mathbf{0},\mathbf{0}) \\ X_{\mathit{Idling}}(t, z{:}\mathbb{R}_{\geq \mathbf{0}}) & = & t \gg \sum_{d:\mathbb{R}_{\geq \mathbf{0}}} (\mathit{int} \cdot Y_{\mathit{Interrupting}}(t+d, \mathbf{0}) \lhd \mathbf{0} \leq 5 \rhd \delta) {}^{\varsigma}(t+d) \\ X_{\mathit{Interrupting}}(t, z{:}\mathbb{R}_{\geq \mathbf{0}}) & = & t \gg \sum_{d:\mathbb{R}_{\geq \mathbf{0}}} [z \leq 5 \land z + d \leq 5] :: \to \\ & & (ret \cdot Y_{\mathit{Idling}}(t+d, \mathbf{0}) \lhd z + d = 5 \rhd \delta) {}^{\varsigma}(t+d) \end{array} \tag{6.4}$$

In a straightforward calculation, we can reduce the complexity of Eqn. (6.4), yielding an equivalent μCRL$_t$-interpretation of the same automaton (see Eqn. (6.5)).

$$\begin{array}{lcl} [\![Y]\!] & = & Y_{\mathit{Idling}}(\mathbf{0}) \\ X_{\mathit{Idling}}(t{:}\mathbb{R}_{\geq \mathbf{0}}) & = & \sum_{d:\mathbb{R}_{\geq \mathbf{0}}} (\mathit{int} \cdot Y_{\mathit{Interrupting}}(t+d)) {}^{\varsigma}(t+d) \\ X_{\mathit{Interrupting}}(t{:}\mathbb{R}_{\geq \mathbf{0}}) & = & (\mathit{ret} \cdot Y_{\mathit{Idling}}(t+5)) {}^{\varsigma}(t+5) \end{array} \tag{6.5}$$

The subsystem, obtained by the parallel composition of the timed automata X and Y is the synchronising timed automaton $X\|_sY$. The μCRL$_t$-interpretation of $X\|_sY$ is the process specified in Eqn. (6.6).

$$[\![X\|_sY]\!] = \rho_R\partial_{\{\mathit{int},\, \mathit{ret}\}}(X_{\mathit{Idling}}(\mathbf{0},\mathbf{0})\|Y_{\mathit{Idling}}(\mathbf{0})) \tag{6.6}$$

The above defined process can be analysed using techniques, developed for μCRL$_t$. For instance, it is possible to analyse the system for deadlocks that may be introduced due to required synchronisations, etc. The interested reader can try linearising the above process, using the expansion and encapsulation theorems of μCRL$_t$ (see e.g. [57]).

6.1.2 Soundness of Interpretation Functions

The translations of Section 6.1.1 are based on a thorough understanding of both timed automata and μCRL$_t$. This, unfortunately, does not guarantee the translations are correct. In other words, the translations might allow us to derive equationally that two timed automata are equivalent in μCRL$_t$ where their original semantics would distinguish their behaviours. In this section, we show the translations preserve the characteristics of a timed automaton.

We start by establishing a relation between the delays and time-stamped executions of actions of a timed automaton and its μCRL$_t$-interpretation. Afterwards, we formulate the soundness result of the interpretations of a timed automaton. Note that we use the results obtained in Chapter 5 allowing us to use the TSLTS semantics of timed automata rather than its TPLTS semantics.

Lemma 6.1.7. Let $X = \langle L, L^0, \Sigma, C, \iota, E\rangle$ be a timed automaton. Then, for all states $(l,\vartheta,t) \in L\times\mathbb{V}_C\times\mathbb{R}_{\geq\mathbf{0}}$, reachable from an initial state of X, Eqn. (6.7) holds.

$$\mathcal{U}_u(l,\vartheta,t) \Leftrightarrow \mathcal{U}_u(X_l(t,\vartheta)) \tag{6.7}$$

Proof. Let (l,ϑ,t) be a reachable state of timed automaton X. We distinguish two cases, viz. $u \leq t$ and $u > t$. Suppose $u \leq t$. Then, an application of Lemma 5.1.22 yields $\mathcal{U}_t(l,\vartheta,t)$. Likewise, from the semantics of μCRL$_t$, we obtain $\mathcal{U}_u(X_l(t,\vartheta))$. The interesting case, therefore is $u > t$. Suppose $u = t + d'$ for some $d' > \mathbf{0}$. Assume $\mathcal{U}_{t+d'}(l,\vartheta,t)$ holds. From $\mathcal{U}_{t+d'}(l,\vartheta,t)$ it follows that we have $\vartheta,\vartheta+d' \models \iota(l)$. Let ν be a valuation for the variable d (found in the interpretation function), such that $\nu(d)$ coincides with the value for d'. This allows us to derive $\vartheta,(\vartheta+d') \models \iota(l)$ is true. This is sufficient to derive $\mathcal{U}_{t+d'}(X_l(t,\vartheta))$.
Conversely, assume $\mathcal{U}_{t+d'}(X_l(t,\vartheta))$ holds. Hence, there is a valuation ν for variable d, such that $\nu(d)$ equals the value for d', $\vartheta,(\vartheta+d') \models \iota(l)$ is true, and $\mathbf{0} < d'$. Therefore, also $\mathcal{U}_{t+d'}(l,\vartheta,t)$.
$\square$

Lemma 6.1.8. Let $X = \langle L, L^0, \Sigma, C, \iota, E\rangle$ be a timed automaton. Then, for all states $(l,\vartheta,t) \in L\times\mathbb{V}_C\times\mathbb{R}_{\geq\mathbf{0}}$, reachable from an initial state of X, and all $d' \in \mathbb{R}_{\geq\mathbf{0}}$ and $\sigma \in \Sigma$, Eqn. (6.8) holds.

$$(l,\vartheta,t) \xrightarrow{\sigma}_{t+d'} (l',\vartheta',t+d') \Leftrightarrow X_l(t,\vartheta) \xrightarrow{\sigma}_{t+d'} X_{l'}(t+d',\vartheta') \tag{6.8}$$

Proof. Let (l, ϑ, t) be a reachable state of the automaton X. Assume $(l, \vartheta, t) \xrightarrow{\sigma}_{t+d'} (l', \vartheta', t + d')$. Then, there must be a switch $l \xrightarrow{\sigma,\varphi,\lambda} l'$, such that $\vartheta, (\vartheta + d') \models \iota(l)$, $(\vartheta + d') \models \varphi$, $\vartheta' = (\vartheta + d')[\lambda := \mathbf{0}]$ and $\vartheta' \models \iota(l')$. Let ν be a valuation for the variables occurring in the location interpretation X_l, such that $\nu(e)$ coincides with $l \xrightarrow{\sigma,\varphi,\lambda} l'$ and $\nu(d)$ coincides with d'. Then, immediately $X_l(t, \vartheta) \xrightarrow{\sigma}_{t+d'} X_{l'}(t + d', \vartheta')$ follows.
Conversely, from $X_l(t, \vartheta) \xrightarrow{\sigma}_{t+d'} X_{l'}(t+d', \vartheta')$ it follows there is a valuation ν for the variables occurring in the location interpretation X_l, such that $\nu(e)$ is assigned some switch $l \xrightarrow{\sigma,\varphi,\lambda} l'$, and $\nu(d)$ coincides with d', for which $\vartheta, (\vartheta + d') \models \iota(l)$, $(\vartheta + d') \models \varphi$, $\vartheta' = (\vartheta + d')[\lambda := \mathbf{0}]$ and $\vartheta' \models \iota(l')$. From this, we immediately have $(l, \vartheta, t) \xrightarrow{\sigma}_{t+d'} (l', \vartheta', t + d')$ □

Theorem 6.1.9 (*Soundness of Interpretation*).
Let $X = \langle L, L^0, \Sigma, C, \iota, E\rangle$ and $Y = \langle L', L'^0, \Sigma', C', \iota', E'\rangle$ be timed automata. Then, Eqn. (6.9) holds.

$$X \leftrightarrow_t Y \;\Leftrightarrow\; [\![X]\!] \leftrightarrow_t [\![Y]\!] \tag{6.9}$$

Proof. We provide witnesses for the timed bisimulation relations and prove these are indeed timed bisimulation relations. Suppose $\mathcal{R}_r{:}X \leftrightarrow_t Y$, where $\mathcal{R}_r$ is a bisimulation relation relating reachable states of X and Y only. Equation (6.10) defines a relation $\mathcal{R}'$.

$$\begin{aligned}\mathcal{R}' = \{(X_l(t, \vartheta), Y_s(u, \chi)) \mid\ & (l, \vartheta, t)\mathcal{R}_r(s, \chi, u), (l, \vartheta, t) \in L \times \mathbb{V}_C \times \mathbb{R}_{\geq \mathbf{0}},\\ & (s, \chi, u) \in L' \times \mathbb{V}_{C'} \times \mathbb{R}_{\geq \mathbf{0}}\}\end{aligned} \tag{6.10}$$

Then, relation $\mathcal{R}'$ defines a timed bisimulation relation, such that $\mathcal{R}'{:}[\![X]\!] \leftrightarrow_t [\![Y]\!]$.

1. Let $X_l(\vartheta, t)\mathcal{R}'Y_s(u, \chi)$. From this, we have $(l, \vartheta, t)\mathcal{R}_r(s, \chi, u)$. Since $\mathcal{R}_r$ is a timed bisimulation relation, relating reachable states, we have $\mathcal{U}_r(l, \vartheta, t)$ iff $\mathcal{U}_r(s, \chi, u)$. An application of Lemma 6.1.7, yields $\mathcal{U}_r(X_l(t, \vartheta))$ iff $\mathcal{U}_r(Y_l(u, \chi))$.

2. Let $X_l(\vartheta, t)\mathcal{R}'Y_s(u, \chi)$. From this, we have $(l, \vartheta, t)\mathcal{R}_r(s, \chi, u)$. Since $\mathcal{R}_r$ is a timed bisimulation, relating the reachable states of X and Y, we can derive that if we have $(l, \vartheta, t) \xrightarrow{\sigma}_r (l', \vartheta', r)$, then also $(s, \chi, u) \xrightarrow{\sigma}_r (s', \chi', r)$ for some $l' \in L$, $s' \in L'$, and $\vartheta', \chi' \in \mathbb{V}_{C\cup C'}$ such that $(l', \vartheta', r)\mathcal{R}_r(s', \chi', r)$. Applying Lemma 6.1.8, then from $X_l(t, \vartheta) \xrightarrow{\sigma}_r X_{l'}(r, \vartheta')$ we can derive $Y_s(u, \chi) \xrightarrow{\sigma}_r Y_{s'}(r, \chi')$ for some $l' \in L$, $s' \in L'$, and $\vartheta', \chi' \in \mathbb{V}_{C\cup C'}$ such that, $X_{l'}(r, \vartheta')\mathcal{R}'Y_{s'}(r, \chi')$. The case where we have a transition from state (s, χ, u) is fully symmetric.

Conversely, suppose $\mathcal{R}{:}[\![X]\!] \leftrightarrow_t [\![Y]\!]$. Equation (6.11) defines a relation $\mathcal{R}''$.

$$\mathcal{R}'' = \{((l, \vartheta, t), (s, \chi, u)) \mid X_l(t, \vartheta)\mathcal{R}Y_s(u, \chi), l \in L, s \in L', \vartheta, \chi \in \mathbb{V}_{C\cup C'}, t, u \in \mathbb{R}_{\geq \mathbf{0}}\} \tag{6.11}$$

Define relation $\mathcal{R}' = \mathcal{R}'' \cap (R_X \times R_Y)$, where R_Z denotes the set of states reachable from an initial state of automaton Z. Then, $\mathcal{R}'$ is a timed bisimulation relation relating timed automata X and Y. Let (l, ϑ, t) and (s, χ, u) denote reachable states of X resp. Y.

1. Let $(l, \vartheta, t)\mathcal{R}'(s, \chi, u)$. Then also $X_l(t, \vartheta)\mathcal{R}Y_s(u, \chi)$. Since $\mathcal{R}$ is a timed bisimulation, we have, if $X_l(t, \vartheta) \xrightarrow{\sigma}_r X_{l'}(r, \vartheta')$ also $Y_s(u, \chi) \xrightarrow{\sigma}_r Y_{s'}(r, \chi')$ for some $l' \in L$, $s' \in L'$, and $\vartheta', \chi' \in \mathbb{V}_{C\cup C'}$ such that $X_{l'}(r, \vartheta')\mathcal{R}Y_{s'}(r, \chi')$. Since (l, ϑ, t) and (s, χ, u) are reachable states of X and Y, we can apply Lemma 6.1.8, and thus we know that if $(l, \vartheta, t) \xrightarrow{\sigma}_r (l', \vartheta', r)$ then also $(s, \chi, u) \xrightarrow{\sigma}_r (s', \chi', r)$ and $(l', \vartheta', r)\mathcal{R}'(s', \chi', r)$. The case where we have a transition from process $Y_s(u, \chi)$ is fully symmetric.

2. Let $(l,\vartheta,t)\mathcal{R}'(s,\chi,u)$. Then also $X_l(t,\vartheta)\mathcal{R}Y_s(u,\chi)$. Relation $\mathcal{R}$ is a timed bisimulation, hence $\mathcal{U}_r(X_l(t,\vartheta))$ iff $\mathcal{U}_r(Y_s(u,\chi))$ for $r\in\mathbb{R}_{\geq\mathbf{0}}$. Since both (l,ϑ,t) and (s,χ,u) are reachable states, applying Lemma 6.1.7, yields $\mathcal{U}_r(l,\vartheta,t)$ iff $\mathcal{U}_r(s,\chi,u)$.

3. Let $l\in L^0$. We have $[\![X]\!]\;\underline{\leftrightarrow}_t[\![Y]\!]$, and therefore $\sum_{l\in L^0} X_l(\mathbf{0},\vartheta_\mathbf{0})\mathcal{R}\sum_{s\in L'^0} Y_l(\mathbf{0},\chi_\mathbf{0})$ for $\vartheta_\mathbf{0}\in\mathbb{V}_C^0$ and $\chi_\mathbf{0}\in\mathbb{V}_{C'}^0$. From this, it follows that also $(l,\vartheta_\mathbf{0},\mathbf{0})\mathcal{R}'(s,\chi_\mathbf{0},\mathbf{0})$ for some $s\in L'^0$. Analogously for $s\in L'^0$.

□

Theorem 6.1.9 establishes a firm relation between the theory of timed automata and the theory of μCRL$_t$. We continue with an investigation of the translation of synchronising timed automata to μCRL$_t$. For this, we first introduce a new μCRL$_t$ operator, abbreviating a standard μCRL$_t$ expression.

Definition 6.1.10. We introduce a binary operator $\gg_\mathbf{0}:\mathbb{R}_{\geq\mathbf{0}}\times\mathbb{P}\to\mathbb{P}$. This operator, is defined by the following equality: $t\gg_\mathbf{0} p=\sum_u p^{\varsigma}u\lhd u\geq t\rhd\delta^{\varsigma}\mathbf{0}$, where p is a process and t is a point in time.

Property 6.1.11 Let t be a point in time and p be a process. Then, we have the following identities:

1. $t\gg_\mathbf{0} t\gg p=t\gg_\mathbf{0} p$,
2. $t\gg_\mathbf{0}(p\|q)=(t\gg_\mathbf{0} p)\|(t\gg_\mathbf{0} q)$.

Proof. We first prove the first property. We can rewrite $t\gg_\mathbf{0} t\gg p$, to the equivalent expression $\sum_u(\sum_v p^{\varsigma}v\lhd v\geq t\rhd\delta^{\varsigma}t)^{\varsigma}u\lhd u\geq t\rhd\delta^{\varsigma}\mathbf{0}$, using the definition of $\gg_\mathbf{0}$ and axioms AT1 and ATB1 through ATB4. The time-stamping operator $^{\varsigma}$ distributes over all other operators. Axiom C4 allows for a further reduction to $\sum_u(\sum_v p^{\varsigma}v^{\varsigma}u\lhd v\geq t\wedge u\geq t\rhd\delta^{\varsigma}t^{\varsigma}u)\lhd u\geq t\rhd\delta^{\varsigma}\mathbf{0}$. Using axiom ATA1, this expression is simplified to $\sum_u p^{\varsigma}u\lhd u\geq t\rhd\delta^{\varsigma}\mathbf{0}$, which, by definition is equal to $t\gg_\mathbf{0} p$.
For the second property, we observe that $t\gg_\mathbf{0}(p\|q)$ can be rewritten to the equivalent expression $\sum_u(p\|q)^{\varsigma}u\lhd u\geq t\rhd\delta^{\varsigma}\mathbf{0}$. Using structural induction, we can prove distributivity of a conditional over parallelism. In combination with the distributivity of $^{\varsigma}$ over all μCRL$_t$ operators, this expression is equivalent to $\sum_u((p^{\varsigma}u\lhd u\geq t\rhd\delta^{\varsigma}\mathbf{0})\|(q^{\varsigma}u\lhd u\geq t\rhd\delta^{\varsigma}\mathbf{0}))$. This expression can be rewritten to $\sum_u((p^{\varsigma}u\lhd u\geq t\rhd\delta^{\varsigma}\mathbf{0})\|\sum_v(q^{\varsigma}v\lhd v\geq t\rhd\delta^{\varsigma}\mathbf{0}))$ using axioms AT1 and ATA1. Repeatedly applying axioms CM1, SUM4, SUM6 and SUM7, yields the equivalent expression $\sum_u(p^{\varsigma}u\lhd u\geq t\rhd\delta^{\varsigma}\mathbf{0})\|\sum_v(q^{\varsigma}v\lhd v\geq t\rhd\delta^{\varsigma}\mathbf{0})$, which, by definition is equivalent to $(t\gg_\mathbf{0} p)\|(t\gg_\mathbf{0} q)$. □

Lemma 6.1.12.
Let $X=\langle L,L^0,\Sigma,C,\iota,E\rangle$ and $Y=\langle L',L'^0,\Sigma',C',\iota',E'\rangle$ be timed automata and let $\Sigma_s=\Sigma\cap\Sigma'$. Then, for all reachable states $(l,\vartheta,t)\|_s(s,\chi,u)$ of the synchronising timed automaton $X\|_sY$, Eqn.(6.12) holds.

$$\mathcal{U}_{t'}((l,\vartheta,t)\|_s(s,\chi,u))\;\Leftrightarrow\;\mathcal{U}_{t'}(\rho_R\partial_{\Sigma_s}(t\max u\gg_\mathbf{0}(X_l(t,\vartheta)\|Y_s(u,\chi))))\tag{6.12}$$

Proof. Let $(l, \vartheta, t)\|_s(s, \chi, u)$ be a reachable state of the synchronising timed automaton $X\|_sY$. Suppose $\mathcal{U}_{t'}((l, \vartheta, t)\|_s(s, \chi, u))$. Then, also $\mathcal{U}_{t'}(l, \vartheta, t)$ and $\mathcal{U}_{t'}(s, \chi, u)$. Since $(l, \vartheta, t)\|_s(s, \chi, u)$ is reachable, also the constituent states (l, ϑ, t) and (s, χ, u) are reachable. An application of Lemma 6.1.7, then also gives $\mathcal{U}_{t'}(X_l(t, \vartheta))$ and $\mathcal{U}_{t'}(Y_s(u, \chi))$. Therefore, we can also derive $\mathcal{U}_{t'}(X_l(t, \vartheta)\|Y_s(u, \chi))$. This is sufficient to derive $\mathcal{U}_{t'}(\rho_R\partial_{\Sigma_s}(t \max u \gg_0 (X_l(t, \vartheta)\|Y_s(u, \chi))))$. For $t' > t \max u$, a proof of the converse is obtained by following the preceding steps in reverse order; for $t \leq t \max u$, Lemma 5.1.26 immediately yields the desired result. □

Lemma 6.1.13.
Let $X = \langle L, L^0, \Sigma, C, \iota, E\rangle$ and $Y = \langle L', L'^0, \Sigma', C', \iota', E'\rangle$ be timed automata. Then for all reachable states $(l, \vartheta, t)\|_s(s, \chi, u)$ of the synchronising timed automaton $X\|_sY$, and all $t' \in \mathbb{R}_{\geq 0}$, and $\sigma \in \Sigma \cap \Sigma' = \Sigma_s$, Eqn.(6.13) holds.

$$\begin{array}{ll} & (l, \vartheta, t)\|_s(s, \chi, u) \xrightarrow{\sigma}_{t'} (l', \vartheta', t')\|_s(s', \chi', t') \\ \Leftrightarrow & \\ & \rho_R\partial_{\Sigma_s}(t \max u \gg_0 (X_l(t, \vartheta)\|Y_s(u, \chi))) \xrightarrow{\sigma}_{t'} \rho_R\partial_{\Sigma_s}(t' \gg_0 (X_{l'}(t', \vartheta')\|Y_{s'}(t', \chi')))\end{array} \tag{6.13}$$

Proof. We use abbreviations for states and location interpretations, e.g. we write $\mathfrak{l}$ for state (l, ϑ, t) and $X_\mathfrak{l}$ for location interpretation $X_l(t, \vartheta)$ of state $\mathfrak{l}$, etc.

Assume $\mathfrak{l}\|_s\mathfrak{s} \xrightarrow{\sigma}_{t'} \mathfrak{l}'\|_s\mathfrak{s}'$. We then have $\mathfrak{l} \xrightarrow{\sigma}_{t'} \mathfrak{l}'$, $\mathfrak{s} \xrightarrow{\sigma}_{t'} \mathfrak{s}'$ and $t' \geq t \max u$. Applying Lemma 6.1.8 immediately yields $X_\mathfrak{l} \xrightarrow{\sigma}_{t'} X_{\mathfrak{l}'}$ and $Y_\mathfrak{s} \xrightarrow{\sigma}_{t'} Y_{\mathfrak{s}'}$. Since both $\mathcal{U}_{t'}(Y_{\mathfrak{s}'})$ and $\mathcal{U}_{t'}(X_{\mathfrak{l}'})$, we have $Y_{\mathfrak{s}'} \mathrel{\underline{\leftrightarrow}}_t t' \gg_0 Y_{\mathfrak{s}'}$ and $X_{\mathfrak{l}'} \mathrel{\underline{\leftrightarrow}}_t t' \gg_0 X_{\mathfrak{l}'}$. Thence, also $X_\mathfrak{l} \xrightarrow{\sigma}_{t'} t' \gg_0 X_{\mathfrak{l}'}$ and $Y_\mathfrak{s} \xrightarrow{\sigma}_{t'} t' \gg_0 Y_{\mathfrak{s}'}$. Since $\mathcal{U}_{t \max u}(X_\mathfrak{l}\|Y_\mathfrak{s})$, also $(X_\mathfrak{l}\|Y_\mathfrak{s}) \mathrel{\underline{\leftrightarrow}}_t t \max u \gg_0 (X_\mathfrak{l}\|Y_\mathfrak{s})$, and hence $t \max u \gg_0 (X_\mathfrak{l}\|Y_\mathfrak{s}) \xrightarrow{\overline{\sigma}}_{t'} t' \gg_0 (X_\mathfrak{l}\|Y_\mathfrak{s})$. Since $R(\overline{\sigma}) = \sigma$, we then immediately also have $\rho_R\partial_{\Sigma_s}(t \max u \gg_0 (X_\mathfrak{l}\|Y_\mathfrak{s})) \xrightarrow{\sigma}_{t'} \rho_R\partial_{\Sigma_s}(t' \gg_0 (X_\mathfrak{l}\|Y_\mathfrak{s}))$.
Conversely, let $\rho_R\partial_{\Sigma_s}(t \max u \gg_0 (X_\mathfrak{l}\|Y_\mathfrak{s})) \xrightarrow{\sigma}_{t'} \rho_R\partial_{\Sigma_s}(t' \gg_0 (X_\mathfrak{l}\|Y_\mathfrak{s}))$. Following the above steps in reverse, we can deduce $t \max u \gg_0 (X_\mathfrak{l}\|Y_\mathfrak{s}) \xrightarrow{\overline{\sigma}}_{t'} t' \gg_0 (X_\mathfrak{l}\|Y_\mathfrak{s})$ and $t \max u \leq t'$ and $X_\mathfrak{l} \xrightarrow{\sigma}_{t'} X_{\mathfrak{l}'}$ and $Y_\mathfrak{s} \xrightarrow{\sigma}_{t'} Y_{\mathfrak{s}'}$. Using Lemma 6.1.8, we then also have $\mathfrak{l} \xrightarrow{\sigma}_{t'} \mathfrak{l}'$ and $\mathfrak{s} \xrightarrow{\sigma}_{t'} \mathfrak{s}'$ and thus also $\mathfrak{l}\|_s\mathfrak{s} \xrightarrow{\sigma}_{t'} \mathfrak{l}'\|_s\mathfrak{s}'$. □

Lemma 6.1.14.
Let $X = \langle L, L^0, \Sigma, C, \iota, E\rangle$ and $Y = \langle L', L'^0, \Sigma', C', \iota', E'\rangle$ be arbitrary timed automata and let $\Sigma_s = \Sigma \cap \Sigma'$. Then for all reachable states $(l, \vartheta, t)\|_s(s, \chi, u)$ of the synchronising timed automaton $X\|_sY$, and all $t' \in \mathbb{T}$, and $\sigma \in \Sigma \setminus \Sigma'$, Eqn.(6.14) holds.

$$\begin{array}{ll} & (l, \vartheta, t)\|_s(s, \chi, u) \xrightarrow{\sigma}_{t'} (l', \vartheta', t')\|_s(s, \chi, u) \\ \Leftrightarrow & \\ & \rho_R\partial_{\Sigma_s}(t \max u \gg_0 (X_l(t, \vartheta)\|Y_s(u, \chi))) \xrightarrow{\sigma}_{t'} \rho_R\partial_{\Sigma_s}(t' \gg_0 X_{l'}(t', \vartheta')\|Y_s(u, \chi))\end{array} \tag{6.14}$$

Proof. We use the convention of abbreviating states and location interpretations as is done in the proof of Lemma 6.1.13.

Assume $\mathfrak{l}\|_s\mathfrak{s} \xrightarrow{\sigma}_{t'} \mathfrak{l}'\|_s\mathfrak{s}$. Then, we also have $\mathfrak{l} \xrightarrow{\sigma}_{t'} \mathfrak{l}'$, $\mathcal{U}_{t'}(\mathfrak{s})$ and $u \leq t'$. Applying Lemma 6.1.8 immediately yields $X_\mathfrak{l} \xrightarrow{\sigma}_{t'} X_{\mathfrak{l}'}$, and Lemma 6.1.7 yields $\mathcal{U}_{t'}(Y_\mathfrak{s})$. Thus, we have $X_\mathfrak{l}\|Y_\mathfrak{s} \xrightarrow{\sigma}_{t'} X_{\mathfrak{l}'}\|(t' \gg Y_\mathfrak{s})$. Since $\mathcal{U}_{t \max u}(X_\mathfrak{l}\|Y_\mathfrak{s})$, we obtain $X_\mathfrak{l}\|Y_\mathfrak{s} \mathrel{\underline{\leftrightarrow}}_t t \max u \gg_0 (X_\mathfrak{l}\|Y_\mathfrak{s})$. Second, since

$\mathcal{U}_{t'}(X_{\mathfrak{l}'}\|(t' \gg Y_{\mathfrak{s}}))$, we have $X_{\mathfrak{l}'}\|(t' \gg Y_{\mathfrak{s}}) \;\underline{\leftrightarrow}\;_t t'\gg_{\mathbf{0}} (X_{\mathfrak{l}'}\|(t' \gg Y_{\mathfrak{s}})) \;\underline{\leftrightarrow}\;_t t'\gg_{\mathbf{0}} (X_{\mathfrak{l}'}\|Y_{\mathfrak{s}})$. Therefore, $t \max u \gg_{\mathbf{0}} (X_{\mathfrak{l}}\|Y_{\mathfrak{s}}) \xrightarrow{\sigma}_{t'} t'\gg_{\mathbf{0}} (X_{\mathfrak{l}'}\|Y_{\mathfrak{s}})$. Thus, $\rho_R\partial_{\Sigma_s}(t \max u \gg_{\mathbf{0}} (X_{\mathfrak{l}}\|Y_{\mathfrak{s}})) \xrightarrow{\sigma}_{t'} \rho_R\partial_{\Sigma_s}(t'\gg_{\mathbf{0}} (X_{\mathfrak{l}'}\|Y_{\mathfrak{s}}))$.
The converse follows from applying the above steps in reverse order. □

Theorem 6.1.15 (*Soundness of Interpretation*).
Let $X = \langle L, L^0, \Sigma, C, \iota, E\rangle$, $Y = \langle L', L'^0, \Sigma', C', \iota', E'\rangle$ and $Z = \langle L'', L''^0, \Sigma \cup \Sigma', C'', \iota'', E''\rangle$ be timed automata, then Eqn. (6.15) holds.

$$X\|_s Y \;\underline{\leftrightarrow}\;_t Z \;\Leftrightarrow\; [\![X\|_s Y]\!] \;\underline{\leftrightarrow}\;_t [\![Z]\!] \tag{6.15}$$

Proof. We again provide witnesses for the bisimulation relations. Let $\Sigma_s = \Sigma \cap \Sigma'$. Suppose $\mathcal{R}_r{:}X\|_s Y \;\underline{\leftrightarrow}\;_t Z$ be a timed bisimulation relation, relating only the reachable states of $X\|_s Y$ to Z and vice versa. Equation (6.16) defines a relation $\mathcal{R}'$.

$$\begin{aligned}\mathcal{R}' = \;& \{(\rho_R\partial_{\Sigma_s}(t \max u \gg_{\mathbf{0}} X_l(t,\vartheta)\|Y_s(u,\chi)), Z_q(v,\zeta)) \mid (l,\vartheta,t)\|_s(s,\chi,u)\mathcal{R}_r(q,\zeta,v),\\ & \vartheta,\chi,\zeta \in \mathbb{V}_{C\cup C'\cup C''}, t,u,v \in \mathbb{R}_{\geq\mathbf{0}}, l \in L, s \in L', q \in L''\}\end{aligned} \tag{6.16}$$

Then, relation $\mathcal{R}'$ is a timed bisimulation up to $\underline{\leftrightarrow}_t$, relating $[\![X\|_s Y]\!]$ to $[\![Z]\!]$.

1. Suppose we have $\rho_R\partial_{\Sigma_s}(t \max u \gg_{\mathbf{0}} X_l(t,\vartheta)\|Y_s(u,\chi))\mathcal{R}' Z_q(v,\zeta)$. From this, we obtain $(l,\vartheta,t)\|_s(s,\chi,u)\mathcal{R}_r(q,\zeta,v)$. Hence, we have, for arbitrary t', $\mathcal{U}_{t'}((l,\vartheta,t)\|_s(s,\chi,u))$ iff $\mathcal{U}_{t'}(q,\zeta,v)$. Applying Lemmas 6.1.12 and 6.1.7, gives $\mathcal{U}_{t'}(t \max u \gg_{\mathbf{0}} X_l(t,\vartheta)\|Y_s(u,\chi))$ iff $\mathcal{U}_{t'}(Z_q(v,\zeta))$.

2. Let $\rho_R\partial_{\Sigma_s}(t \max u \gg_{\mathbf{0}} X_l(t,\vartheta)\|Y_s(u,\chi))\mathcal{R}' Z_q(v,\zeta)$, and assume $\sigma \in \Sigma_s$. We again have $(l,\vartheta,t)\|_s(s,\chi,u)\mathcal{R}_r(q,\zeta,v)$. Since relation $\mathcal{R}_r$ is a timed bisimulation up to $\underline{\leftrightarrow}_t$, we know when $(l,\vartheta,t)\|_s(s,\chi,u) \xrightarrow{\sigma}_{t'} (l',\vartheta',t')\|_s(s',\chi',t')$ then also $(q,\zeta,v) \xrightarrow{\sigma}_{t'} (q',\zeta',t')$ and again $(l',\vartheta',t')\|_s(s',\chi',t')\mathcal{R}_r(q',\zeta',t')$. By Lemmas 6.1.13 and 6.1.8, we then obtain if $\rho_R\partial_{\Sigma_s}(t \max u \gg_{\mathbf{0}} (X_l(t,\vartheta)\|Y_s(u,\chi))) \xrightarrow{\sigma}_{t'} \rho_R\partial_{\Sigma_s}(t'\gg_{\mathbf{0}} (X_{l'}(t',\vartheta')\|Y_{s'}(t',\chi')))$ then also $Z_q(v,\zeta) \xrightarrow{\sigma}_{t'} Z_{q'}(t',\zeta')$ and $\rho_R\partial_{\Sigma_s}(t'\gg_{\mathbf{0}} (X_{l'}(t',\vartheta')\|Y_{s'}(t',\chi')))\mathcal{R}' Z_{q'}(t',\zeta')$. The case where we have a transition from state (q,ζ,v) is symmetric.

   Assume $\sigma \in \Sigma \setminus \Sigma'$. Again, we have $(l,\vartheta,t)\|_s(s,\chi,u)\mathcal{R}_r(q,\zeta,v)$, and since $\mathcal{R}_r$ is a timed bisimulation up to $\underline{\leftrightarrow}_t$, we have if $(l,\vartheta,t)\|_s(s,\chi,u) \xrightarrow{\sigma}_{t'} (l',\vartheta',t')\|_s(s,\chi,u)$ then also $(q,\zeta,v) \xrightarrow{\sigma}_{t'} (q',\zeta',t')$ and again $(l',\vartheta',t')\|_s(s,\chi,u)\mathcal{R}_r(q',\zeta',t')$. Hence, we also have if $\rho_R\partial_{\Sigma_s}(t \max u \gg_{\mathbf{0}} (X_l(t,\vartheta)\|Y_s(u,\chi))) \xrightarrow{\sigma}_{t'} \rho_R\partial_{\Sigma_s}(t'\gg_{\mathbf{0}} (X_{l'}(t',\vartheta')\|Y_s(u,\chi)))$ then also $Z_q(v,\zeta) \xrightarrow{\sigma}_{t'} Z_{q'}(t',\zeta')$ and $\rho_R\partial_{\Sigma_s}(t'\gg_{\mathbf{0}} (X_{l'}(t',\vartheta')\|Y_s(u,\chi)))\mathcal{R}' Z_{q'}(t',\zeta')$ by Lemmas 6.1.14 and 6.1.8. Similarly, when we have a transition from state (q,ζ,v), and when we have an action $\sigma \in \Sigma' \setminus \Sigma$.

Conversely, let $\mathcal{R}{:}[\![X\|_s Y]\!] \;\underline{\leftrightarrow}\;_t [\![Z]\!]$. Equation (6.17) defines the relation $\mathcal{R}''$.

$$\begin{aligned}\mathcal{R}'' = \;& \{((l,\vartheta,t)\|_s(s,\chi,u),(q,\zeta,v)) \mid \rho_R\partial_{\Sigma_s}(t \max u \gg_{\mathbf{0}} X_l(t,\vartheta)\|Y_s(u,\chi))\mathcal{R} Z_q(v,\zeta),\\ & \vartheta,\chi,\zeta \in \mathbb{V}_{C\cup C'\cup C''}, t,u,v \in \mathbb{R}_{\geq\mathbf{0}}, l \in L, s \in L', q \in L''\}\end{aligned} \tag{6.17}$$

Relation $\mathcal{R}' = \mathcal{R}'' \cap (R_{X\|_s Y} \times R_Z)$ is a timed bisimulation up to $\underline{\leftrightarrow}_t$, relating $X\|_s Y$ and Z; here R_V is the set of all reachable states of automaton V.

1. Suppose $(l, \vartheta, t)\|_s(s, \chi, u)\mathcal{R}'(q, \zeta, v)$. Thence, we can immediately infer the relation between the location interpretations, i.e. $\rho_R\partial_{\Sigma_s}(t \max u \gg_{\mathbf{0}} X_l(t, \vartheta)\|Y_s(u, \chi))\mathcal{R}Z_q(v, \zeta)$, and since $\mathcal{R}$ is a bisimulation up to $\leftrightarrow_t$, also if $\mathcal{U}_{t'}(\rho_R\partial_{\Sigma_s}(t \max u \gg_{\mathbf{0}} X_l(t, \vartheta)\|Y_s(u, \chi)))$ then $\mathcal{U}_{t'}(Z_q(v, \zeta))$. By Lemmas 6.1.12 and 6.1.7, we then also have if $\mathcal{U}_{t'}((l, \vartheta, t)\|_s(s, \chi, u))$ then $\mathcal{U}_{t'}(q, \zeta, v)$. The case where we have a transition from process $Z_q(v, \zeta)$ is symmetric.

2. Suppose $(l, \vartheta, t)\|_s(s, \chi, u)\mathcal{R}'(q, \zeta, v)$, and suppose $\sigma \in \Sigma_s$. Via relation $\mathcal{R}'$, we again obtain $\rho_R\partial_{\Sigma_s}(t \max u \gg_{\mathbf{0}} X_l(t, \vartheta)\|Y_s(u, \chi))\mathcal{R}Z_q(v, \zeta)$, and since $\mathcal{R}$ is a timed bisimulation, we have if $\rho_R\partial_{\Sigma_s}(t \max u \gg_{\mathbf{0}} X_l(t, \vartheta)\|Y_s(u, \chi)) \xrightarrow{\sigma}_{t'} \rho_R\partial_{\Sigma_s}(t' \gg_{\mathbf{0}} X_{l'}(t', \vartheta')\|Y_{s'}(t', \chi'))$ then also $Z_q(v, \zeta) \xrightarrow{\sigma}_{t'} Z_{q'}(t', \zeta')$ and $\rho_R\partial_{\Sigma_s}(t' \gg_{\mathbf{0}} X_{l'}(t', \vartheta')\|Y_{s'}(t', \chi'))\mathcal{R}Z_{q'}(t', \zeta')$. Applying Lemmas 6.1.13 and 6.1.8, yields if $(l, \vartheta, t)\|_s(s, \chi, u) \xrightarrow{\sigma}_{t'} (l', \vartheta', t')\|_s(s', \chi', t')$ then also $(q, \zeta, v) \xrightarrow{\sigma}_{t'} (q', \zeta', t')$ such that $(l', \vartheta', t')\|_s(s', \chi', t')\mathcal{R}'(q', \zeta', t')$. Similarly, Lemmas 6.1.14 and 6.1.8 yield the required results in case of actions $\sigma \in (\Sigma\backslash\Sigma')\cup(\Sigma'\backslash\Sigma)$. The case where we have a transition from the process $Z_q(v, \zeta)$ is symmetric.

3. Let $l \in L^0$ and $s \in L'^0$. Since we already know $\mathcal{R}{:}[\![X\|_s Y]\!] \leftrightarrow_t [\![Z]\!]$, we must surely have $\partial_R\rho_{\Sigma_s}((\sum_{l\in L^0} X_l(\mathbf{0}, \vartheta_{\mathbf{0}}))\|(\sum_{s\in L'^0} Y_s(\mathbf{0}, \chi_{\mathbf{0}})))\mathcal{R}(\sum_{q\in L''^0} Z_q(\mathbf{0}, \zeta_{\mathbf{0}}))$, for $\vartheta_{\mathbf{0}} \in \mathbb{V}^0_C$, $\chi_{\mathbf{0}} \in \mathbb{V}^0_{C'}$ and $\zeta_{\mathbf{0}} \in \mathbb{V}^0_{C''}$. Hence, it also follows that $(l, \vartheta_{\mathbf{0}}, \mathbf{0})\|_s(s, \chi_{\mathbf{0}}, \mathbf{0})\mathcal{R}'(q, \zeta_{\mathbf{0}}, \mathbf{0})$ for some $q \in L''^0$. Analogously for $q \in L''^0$.

□

6.1.3 Summary

Summarising, the interpretations of timed automata and synchronising timed automata, defined in Section 6.1.1 are sound. Consequently, we can employ the equational theory of μCRL$_t$ to reason about timed automata. This can, for example, be useful to construct smaller μCRL$_t$ models using rewrite techniques also employed for μCRL and μCRL$_t$.

Several remarks are in order. In Chapter 5, we considered the subclass of timed automata that are "sensible", in the sense that their initial locations can indeed function as a starting state of the system, without causing the timed automaton to have *deadlocked*. The above results therefore apply to sensible timed automata only. Timed automata that are not sensible, cannot be translated to μCRL$_t$, mainly because of the absence of a process representing a *deadlocked* situation. In [125], this is mended by extending μCRL$_t$ with an *immediate deadlock* $\dot{\delta}$.

Second, we observe that there is, in general, no inverse translation from μCRL$_t$ to an equivalent timed automaton, as the example process of Eqn. (6.18) illustrates. Thence, μCRL$_t$ is strictly more expressive than (sensible) timed automata.

$$X(t{:}\mathbb{R}_{\geq\mathbf{0}}) = \sum_v \textit{run}^{\varsigma} v \cdot \textit{stop}^{\varsigma} 2v \cdot X(2v) \tag{6.18}$$

It is well conceivable that a more liberal schema of clock constraints gives rise to a more expressive formalism (for an impression of the expressivity of various extensions of "standard" timed automata see e.g. *stopwatch automata*[34], updatable timed automata [28]). For practical purposes, however, there is not much that can be gained, as the timed automata, as discussed in

Chapters 5 and 6 seem to have already stretched decidability to its limits (see also the discussion below).

Finally, the results obtained in this section imply that there is a class of μCRL$_t$ processes for which model checking is a viable option. Added to the extra expressive power due to the presence of data, μCRL$_t$ has the potential to become a very powerful tool for verifying timed processes.

Bibliographic Notes

The topic of expressivity and decidability of various properties for (extensions of) timed automata is a blossoming area of research. In the early 1990s, Alur and Dill [7], showed that the language-emptiness problem and timed bisimulation for timed automata are decidable, but that the language-universality problem (see also [66]) and the language inclusion are undecidable. Recently, it is shown that the language-universality problem for a subclass of timed automata, called *open* timed automata, is decidable if the dense time domain is "weakly monotonic", i.e. when several events are allowed at the same time (see [100]). In [28], it was proved that decidability can also be preserved when allowing more general update operations, i.e. reinitialisations of clocks other than to $\mathbf{0}$. The *restricted additive clock constraints* (clock constraints of the form $(c_1 + c_2) \sim v$, where c_1 and c_2 are clocks), we allow in the timed automata we considered in Chapter 5 have only recently been shown to yield decidability of the emptiness problem [19]. To give an impression of several extensions of timed automata for which the reachability property is undecidable (see also [64]), we list a number of well-known examples:

- additive clock constraints, i.e. clock constraints of the form $(c_1 + c_2) \sim (c_1' + c_2')$ [8],
- two non-clock constant-slope variables [6],
- three stopwatches [35],
- one memory cell and assignments between variables [35].

6.2 Interpreting Hybrid Automata

The interpretation functions for hybrid automata and synchronising hybrid automata are defined in Section 6.2.1. Their correctness is assessed in Section 6.2.2. We summarise the results in Section 6.2.3.

6.2.1 Interpretation Functions

Timed Automata and hybrid automata do not differ too much in their syntax. It may therefore not come as a surprise that the translation of a hybrid automaton to a μCRL$_t$ process follows roughly the same lines as the translation of timed automata to μCRL$_t$ processes. We start by introducing additional notation.

Notation 6.2.1. For a hybrid automaton $X = \langle M, M^0, \Sigma, V, \iota, \theta, \mathcal{I}, E\rangle$, we write E_m for the set of switches originating from control mode m. If e is a switch in E_m, then by φ_e we mean the guard of switch e, by σ_e we mean the action that is executed upon taking switch e, by ρ_e we mean the resets of e and by m_e, we mean the target control mode of e.

Definition 6.2.2 (*μCRL$_t$-Interpretation of a Hybrid Automaton, Control Mode Interpretation*). Let $X = \langle M, M^0, \Sigma, V, \iota, \theta, \mathcal{I}, E\rangle$ be a hybrid automaton. The μCRL$_t$-interpretation $[\![X]\!]{:}\mathbb{P}$ is defined as $[\![X]\!] = \sum_{m\in M^0}\sum_{\vartheta\in\mathbb{V}_V} [\vartheta \models \mathcal{I}(m)] :: \rightarrow X_m(\mathbf{0}, \vartheta)$, where X_m for all $m \in M$ is defined by Eqn. (6.19).

$$\begin{aligned} X_m = \;& \lambda t{:}\mathbb{R}_{\geq\mathbf{0}}.\lambda\vartheta{:}\mathbb{V}_V.(t \gg \textstyle\sum_{d:\mathbb{R}_{\geq\mathbf{0}}}\sum_{f:\mathcal{D}_d} \\ & [f \models \iota(m) \wedge f(\mathbf{0}) = \vartheta \wedge f\rceil\omega_f \models \theta(m)] :: \rightarrow \textstyle\sum_{e\in E_m}\sum_{\vartheta':\mathbb{V}_{V'}} \\ & (\sigma_e \cdot X_{m_e}(t+d, \vartheta') \lhd f(d) \models \varphi_e \wedge \vartheta' \models \rho_e[f(d)] \wedge \vartheta' \models \iota(m_e) \rhd \delta\;)^{\varsigma}(t+d)) \end{aligned} \tag{6.19}$$

Here, we use the set $\mathcal{D}_d$ to denote the set $\mathcal{D}_d^{\mathbb{V}_V}$ defined in the previous chapter. For a hybrid automaton X and a control mode m of this automaton X, the process X_m is called the *control mode interpretation*.

Even though the μCRL$_t$-interpretation of a timed automaton resembles the μCRL$_t$-interpretation of a hybrid automaton, the apparent complexity of the latter is much higher, due to the extra summands for the witness functions and the reset-valuations. A translated hybrid automaton, therefore, is quite complex. The following example demonstrates the results of the translation of a relatively simple hybrid automaton.

Example 6.2.3. We again consider the thermostat example (see also Example 5.2.20). Let X be the hybrid automaton depicted in Fig.6.4, representing the thermostat. The μCRL$_t$-interpretation

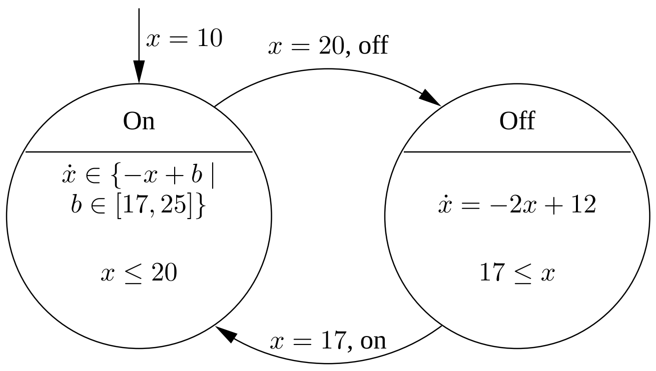


Figure 6.4: Thermostat Model

of automaton X is given in Eqn. (6.20). It consists of two control mode interpretations, viz. X_{On} and X_{Off}, and an initialisation, represented by process $[\![X]\!]$. The witness functions in both control mode interpretations are represented by variable f.

$$\begin{array}{lcl}
[\![X]\!] & = & X_{\mathit{On}}(\mathbf{0}, 10) \\
X_{\mathit{On}}(t{:}\mathbb{R}_{\geq \mathbf{0}}, x_c{:}\mathbb{R}) & = & t \gg \sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \sum_{f:\mathcal{D}_d} \\
 & & [f \models (x \leq 20) \wedge f(\mathbf{0})(x) = x_c \wedge \\
 & & f\rceil\omega_f \models (\dot{x} \in \{-x + b \mid b \in [17, 25]\})] :: \!\!\rightarrow \\
 & & (\mathit{on} \cdot X_{\mathit{Off}}(t + d, f(d)(x)) \\
 & & \;\lhd f(d)(x) = 20 \wedge f(d)(x) \models (x \geq 17) \rhd \delta) {}^{\varsigma}(t + d) \\
X_{\mathit{Off}}(t{:}\mathbb{R}_{\geq \mathbf{0}}, x_c{:}\mathbb{R}) & = & t \gg \sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \sum_{f:\mathcal{D}_d} \\
 & & [f \models (x \geq 17) \wedge f(\mathbf{0})(x) = x_c \wedge f\rceil\omega_f \models (\dot{x} = -2x + 12)] :: \!\!\rightarrow \\
 & & (\mathit{on} \cdot X_{\mathit{On}}(t + d, f(d)(x)) \\
 & & \;\lhd f(d)(x) = 17 \wedge f(d)(x) \models (x \leq 20) \rhd \delta) {}^{\varsigma}(t + d)
\end{array} \tag{6.20}$$

Although the above equations seem complex, some elementary calculations using the μCRL$_t$ axiom system allows for a surprising reduction of the complexity (see Eqn. (6.21)). For instance, we can infer that $x_c \leq 17$ is an invariant for location-interpretation X_{On}. This invariant allows us to find the first moment the thermostat reaches 20° Celsius by solving the above differential equation.

$$\begin{array}{lcl}
[\![X]\!] & = & X_{\mathit{On}}(\mathbf{0}, 10) \\
X_{\mathit{On}}(t{:}\mathbb{R}_{\geq \mathbf{0}}, x_c{:}\mathbb{R}) & = & \sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \mathit{off} {}^{\varsigma}(t + d) \cdot X_{\mathit{Off}}(t + d, 20) \lhd d \geq \ln \frac{25 - x_c}{5} \rhd \delta {}^{\varsigma}\mathbf{0} \\
X_{\mathit{Off}}(t{:}\mathbb{R}_{\geq \mathbf{0}}, x_c{:}\mathbb{R}) & = & \mathit{on} {}^{\varsigma}(t + \frac{1}{2} \ln \frac{14}{11}) \cdot X_{\mathit{On}}(t + \frac{1}{2} \ln \frac{14}{11}, 17)
\end{array} \tag{6.21}$$

The translation of synchronising hybrid automata matches the translation of synchronising timed automata. This is indeed what is to be expected, as the semantics for a synchronising timed automaton matches that of a synchronising hybrid automaton. For completeness' sake, we repeat the translation.

Definition 6.2.4 (*μCRL$_t$-Interpretation of Synchronising Hybrid Automata*).
Let $X = \langle M, M^0, \Sigma, V, \iota, \theta, \mathcal{I}, E \rangle$ and $Y = \langle M', M'^0, \Sigma', V', \iota', \theta', \mathcal{I}', E' \rangle$ be hybrid automata. The μCRL$_t$-interpretation of $X \|_s Y$ is defined by Eqn. (6.22).

$$[\![X \|_s Y]\!] = \rho_R(\partial_{\Sigma \cap \Sigma'}([\![X]\!] \| [\![Y]\!])) \tag{6.22}$$

We define the set $\overline{\Sigma} \cap \overline{\Sigma}' = \{\overline{\sigma} \mid \sigma \in \Sigma \cap \Sigma'\}$; we define a function $R{:}\overline{\Sigma} \cap \overline{\Sigma}' \rightarrow (\Sigma \cap \Sigma')$ as $R(\overline{\sigma}) = \sigma$. The communication function is defined as $\gamma(\sigma, \sigma) = \overline{\sigma}$ and δ otherwise.

6.2.2 Soundness of Interpretation Functions

In this section, we assess the soundness of the translations defined in Section 6.2.1. We first relate our μCRL$_t$-interpretations to hybrid automata interpreted in the alternative semantics. On the basis of the results of Chapter 5, we can then draw our conclusions for the relation between our μCRL$_t$-interpretations and hybrid automata interpreted in the standard semantics.

We start by proving two technical lemmata, establishing a relation between a hybrid automaton and its μCRL$_t$-interpretation.

Lemma 6.2.5. Let $X = \langle M, M^0, \Sigma, V, \iota, \theta, \mathcal{I}, E\rangle$ be a hybrid automaton. Then, for all reachable states $(m, \vartheta, t) \in M \times \mathbb{V}_V \times \mathbb{R}_{\geq \mathbf{0}}$, and all points in time $u \in \mathbb{R}_{\geq \mathbf{0}}$, Eqn. (6.23) holds.

$$\mathcal{U}_u(m, \vartheta, t) \ \Leftrightarrow\ \mathcal{U}_u(X_m(t, \vartheta)) \tag{6.23}$$

Proof. We distinguish the cases $u \leq t$ and $u = t + d'$ with $d' > \mathbf{0}$. The case where $u \leq t$ holds by the grace of Lemma 5.2.33 and the semantics of μCRL$_t$.
Assume $u = t + d'$ for some $d' > \mathbf{0}$. Assume $\mathcal{U}_{t+d'}(m, \vartheta, t)$. Thus, we know there exists a continuous function $F{:}[\mathbf{0}, d'] \to \mathbb{V}_V$ such that $\mathsf{diff}(F)$ a.e., $F(\mathbf{0}) = \vartheta$, $F \models \iota(m)$ and $F\rceil\omega_F \models \theta(m)$. Let ν be a valuation for the variables f and d, occurring in the control mode interpretation X_m, such that $\nu(f)$ coincides with F and $\nu(d)$ equals the value for d'. Then, we know $f \models \iota(m) \wedge f(\mathbf{0}) = \vartheta \wedge f\rceil\omega_f \models \theta(m)$ evaluates to true under valuation ν. Thus, also $\mathcal{U}_{t+d'}(X_m(t, \vartheta))$ holds.
Conversely, assume $\mathcal{U}_{t+d'}(X_m(t, \vartheta))$ holds. Then, there is a valuation ν for variables d' and f, such that $\nu(d)$ equals the value for d', and the function $\nu(f)$ is such that $f \models \iota(m) \wedge f(\mathbf{0}) = \vartheta \wedge f\rceil\omega_f \models \theta(m)$ is true under valuation ν. Thence, also $\mathcal{U}_{t+d'}(l, \vartheta, t)$. □

Lemma 6.2.6. Let $X = \langle M, M^0, \Sigma, V, \iota, \theta, \mathcal{I}, E\rangle$ be a hybrid automaton. Then, for all states $(m, \vartheta, t) \in M \times \mathbb{V}_V \times \mathbb{R}_{\geq \mathbf{0}}$, reachable from an initial state of X, and for all $d' \in \mathbb{R}_{\geq \mathbf{0}}$, Eqn. (6.24) holds.

$$(m, \vartheta, t) \xrightarrow{\sigma}_{t+d'} (m', \vartheta'', t + d') \ \Leftrightarrow\ X_m(t, \vartheta) \xrightarrow{\sigma}_{t+d'} X_{m'}(t + d', \vartheta'') \tag{6.24}$$

Proof. Assume $(m, \vartheta, t) \xrightarrow{\sigma}_{t+d'} (m', \vartheta'', t + d')$. Then, there must exist a switch $m \xrightarrow{\sigma,\varphi,\rho} m'$ and a continuous function $F{:}[\mathbf{0}, d'] \to \mathbb{V}_V$, such that $F(\mathbf{0}) = \vartheta$, $F \models \iota(m)$, $F\rceil\omega_F \models \theta(m)$, $F(d') \models \varphi$, $\vartheta' \models \rho[F(d')]$ and $\vartheta'' \models \iota(m')$ are satisfied. Hence, there exists a valuation ν for variables d, e, f and ϑ' such that $\nu(d)$ equals d', $\nu(e)$ equals $m \xrightarrow{\sigma,\varphi,\rho} m'$, $\nu(f)$ coincides with F and $\nu(\vartheta')$ is ϑ''. Then, we know $f \models \iota(m) \wedge f(\mathbf{0}) = \vartheta \wedge f\rceil\omega_f \models \theta(m)$ under valuation ν is true, and, moreover, $f(d) \models \varphi_e \wedge \vartheta' \models \rho_e[f(d)] \wedge \vartheta' \models \iota(m_e)$ also evaluates to true under valuation ν. Then, we have $X_m(t, \vartheta) \xrightarrow{\sigma}_{t+d'} X_{m'}(t + d', \vartheta'')$.
Conversely, from $X_m(t, \vartheta) \xrightarrow{\sigma}_{t+d'} X_{m'}(t{+}d', \vartheta'')$, it follows there is a valuation ν, for which we know there is a switch $\nu(e)$ equal to $m \xrightarrow{\sigma,\varphi,\rho} m'$, such that $f \models \iota(m) \wedge f(\mathbf{0}) = \vartheta \wedge f\rceil\omega_f \models \theta(m)$ and $f(d) \models \varphi \wedge \vartheta' \models \rho[f(d)] \wedge \vartheta' \models \iota(m')$ both hold for this particular valuation of the variables d, e, f and ϑ'. Thus, we have $\nu(f){:}[\mathbf{0}, \nu(d)] \to \mathbb{V}_V$, where $\nu(d)$ coincides with d', and $\nu(\vartheta')$ coincides with ϑ'', satisfying the hypothesis of the action deduction rule of a hybrid automaton. Hence, we can derive $(m, \vartheta, t) \xrightarrow{\sigma}_{t+d'} (m', \vartheta'', t + d')$. □

Theorem 6.2.7 (*Soundness of Interpretation*).
Let $X = \langle M, M^0, \Sigma, V, \iota, \theta, \mathcal{I}, E\rangle$ and $Y = \langle M', M'^0, \Sigma', V', \iota', \theta', \mathcal{I}', E'\rangle$ be arbitrary hybrid automata. Then, Eqn. (6.25) holds.

$$X \leftrightarroweq_t Y \ \Leftrightarrow\ [\![X]\!] \leftrightarroweq_t [\![Y]\!] \tag{6.25}$$

Proof. We construct the bisimulation relations witnessing the equality of Eqn. (6.25). Suppose $\mathcal{R}_r{:}X \leftrightarroweq_t Y$, where $\mathcal{R}_r$ is a timed bisimulation relation relating only the reachable states of X and Y. Equation (6.26) defines a binary relation $\mathcal{R}'$.

$$\begin{aligned}\mathcal{R}' = \{(X_m(t, \vartheta), Y_n(u, \chi)) \mid\ & (m, \vartheta, t)\mathcal{R}_r(n, \chi, u), (m, \vartheta, t) \in L \times \mathbb{V}_V \times \mathbb{R}_{\geq \mathbf{0}},\\ & (n, \chi, u) \in L' \times \mathbb{V}_{V'} \times \mathbb{R}_{\geq \mathbf{0}}\}\end{aligned} \tag{6.26}$$

Relation $\mathcal{R}'$ defines a timed bisimulation relation, such that $\mathcal{R}':[\![X]\!] \leftrightarrow_t [\![Y]\!]_t$. The proof follows the same line of reasoning as the proof of Theorem 6.1.9.

Conversely, assume $\mathcal{R}:[\![X]\!] \leftrightarrow_t [\![Y]\!]$. Let $\mathcal{R}''$ be the relation, defined by Eqn. (6.27).

$$\mathcal{R}' = \{((m,\vartheta,t),(n,\chi,u)) \mid X_m(t,\vartheta)\mathcal{R}Y_n(u,\chi), \vartheta,\chi \in \mathbb{V}_{V\cup V'}, t,u \in \mathbb{R}_{\geq \mathbf{0}}, m \in M, n \in N'\} \tag{6.27}$$

Define $\mathcal{R}' = \mathcal{R}'' \cap (R_X \times R_Y)$, where R_Z is exactly the set of reachable states of automaton Z. Then, relation $\mathcal{R}'$ is a timed bisimulation relation, such that $\mathcal{R}':X \leftrightarrow_t Y$. The proof thereof proceeds along the lines of the proof of Theorem 6.1.9. □

Based on the observations of Chapter 5, we now state the correspondence theorem for hybrid automata, interpreted in the standard semantics and their μCRL$_t$-interpretations.

Theorem 6.2.8 (*Approximable Hybrid Automata and μCRL$_t$*).
For all approximable hybrid automata X and Y, Eqn. (6.28) holds.

$$X \leftrightarrow_t^{\iota} Y \;\Leftrightarrow\; [\![X]\!] \leftrightarrow_t [\![Y]\!] \tag{6.28}$$

Proof. Follows immediately from the combination of Theorems 5.2.48 and 6.2.7 □

The following technical lemmata establish the connection between a synchronising hybrid automaton and its μCRL$_t$-interpretation. The results and proofs coincide with the results and proofs of Lemmata 6.1.12 through 6.1.14; we therefore omit the proofs.

Lemma 6.2.9. Let $X = \langle M, M^0, \Sigma, V, \iota, \theta, \mathcal{I}, E\rangle$ and $Y = \langle M', M'^0, \Sigma', V', \iota', \theta', \mathcal{I}', E'\rangle$ be hybrid automata and let $\Sigma_s = \Sigma \cap \Sigma'$. Then, for all reachable states $(m,\vartheta,t)\|_s(n,\chi,u)$, and all $t' \in \mathbb{R}_{\geq \mathbf{0}}$, Eqn. (6.29) holds.

$$\mathcal{U}_{t'}((m,\vartheta,t)\|_s(n,\chi,u)) \;\Leftrightarrow\; \mathcal{U}_{t'}(\rho_R\partial_{\Sigma_s}(t \max u \gg_{\mathbf{0}} (X_m(t,\vartheta)\|Y_n(u,\chi)))) \tag{6.29}$$

Proof. Omitted. □

Lemma 6.2.10. Let $X = \langle M, M^0, \Sigma, V, \iota, \theta, \mathcal{I}, E\rangle$ and $Y = \langle M', M'^0, \Sigma', V', \iota', \theta', \mathcal{I}', E'\rangle$ be hybrid automata and let $\Sigma_s = \Sigma \cap \Sigma'$. Then, for all reachable states $(m,\vartheta,t)\|_s(n,\chi,u)$, all $t' \in \mathbb{R}_{\geq \mathbf{0}}$, and all $\sigma \in \Sigma_s$, Eqn. (6.30) holds.

$$\begin{array}{ll} & (m,\vartheta,t)\|_s(n,\chi,u) \xrightarrow{\sigma}_{t'} (m',\vartheta',t')\|_s(n',\chi',t') \\ \Leftrightarrow & \\ & \rho_R\partial_{\Sigma_s}(t \max u \gg_{\mathbf{0}} (X_m(t,\vartheta)\|Y_n(u,\chi))) \xrightarrow{\sigma}_{t'} \rho_R\partial_{\Sigma_s}(t' \gg_{\mathbf{0}} (X_{m'}(t',\vartheta')\|Y_{n'}(t',\chi')))\end{array} \tag{6.30}$$

Proof. Omitted. □

Lemma 6.2.11. Let $X = \langle M, M^0, \Sigma, V, \iota, \theta, \mathcal{I}, E\rangle$ and $Y = \langle M', M'^0, \Sigma', V', \iota', \theta', \mathcal{I}', E'\rangle$ be hybrid automata and let $\Sigma_s = \Sigma \cap \Sigma'$. Then, for all reachable states $(m, \vartheta, t)\|_s(n, \chi, u)$, all $t' \in \mathbb{R}_{\geq \mathbf{0}}$, and all $\sigma \in \Sigma \setminus \Sigma'$, Eqn. (6.31) holds.

$$\begin{aligned} & (m, \vartheta, t)\|_s(n, \chi, u) \xrightarrow{\sigma}_{t'} (m', \vartheta', t')\|_s(n, \chi, u) \\ \Leftrightarrow & \\ & \rho_R\partial_{\Sigma_s}(t \max u \gg_{\mathbf{0}} (X_m(t, \vartheta)\|Y_n(u, \chi))) \xrightarrow{\sigma}_{t'} \rho_R\partial_{\Sigma_s}(t' \gg_{\mathbf{0}} (X_{m'}(t', \vartheta')\|Y_n(u, \chi))) \end{aligned} \tag{6.31}$$

Proof. Omitted. □

Theorem 6.2.12 (*Soundness of Interpretation*).
Assume the hybrid automata $X = \langle M, M^0, \Sigma, V, \iota, \theta, \mathcal{I}, E\rangle$, $Y = \langle M', M'^0, \Sigma', V', \iota', \theta', \mathcal{I}', E'\rangle$ and $Z = \langle M'', M''^0, \Sigma'', V'', \iota'', \theta'', \mathcal{I}'', E''\rangle$. Then Eqn. (6.32) holds.

$$X\|_sY \leftrightarroweq_t Z \;\Leftrightarrow\; [\![X\|_sY]\!] \leftrightarroweq_t [\![Z]\!] \tag{6.32}$$

Proof. Omitted. □

6.2.3 Summary

We can draw several conclusions from the preceding sections. The interpretations of hybrid automata and synchronising hybrid automata, defined in Section 6.2.1 are sound with respect to the alternative semantics of hybrid automata (see Chapter 5). However, these soundness results are true only for a class of hybrid automata (i.e. approximable hybrid automata), when we consider the standard semantics of hybrid automata. This implies that in general, the systems specified as hybrid automata are incomparable to the systems specified in μCRL$_t$. It also means the class of hybrid systems that can be modelled by approximable hybrid automata may take a special place in the field of hybrid systems. It may prove worth it to pursue a more in-depth investigation into this class.

Slightly worrying is the reason for the incompatibility of μCRL$_t$ and the framework of hybrid automata. As we mentioned at the start of this chapter, time as an entity is well studied in real-time discrete systems. There, time additivity is assumed to be a basic property for most real-time formalisms (save a few). On the other hand, the implications of assuming time additivity for continuous systems are not yet fully clear. However, *concatenation of solutions*, an accepted property in the realm of continuous systems, is likely to coincide with time additivity for hybrid systems. In any case, it is imaginable, but not likely, that insisting on time additivity can compromise the theory that has been developed for continuous systems. In this case, a new investigation into the properties of time is needed. In any other case, the absence of the time additivity property in hybrid automata is wanting, and the standard semantics of hybrid automata must be considered flawed.

More on a practical note, the class of approximable hybrid automata covers all hybrid automata for which automated verification is feasible to date. This means we can identify several classes of μCRL$_t$ expressions, representing hybrid systems, for which model-checking tool support can be built. The construction of such tools, however, is no easy task.

Chapter 7

Conveyor Belt — Case Study

This final chapter of this thesis is devoted to a case study in μCRL$_t$. In the past, many case studies have been conducted in μCRL (see e.g. [74, 55, 112]). These studies have signalled various bottle-necks in μCRL and, at the same time, identified numerous strong-points of μCRL. Some of the identified bottle-necks have led to the investigation of novel techniques, culminating into e.g. Cones and Foci [54], and advanced model-checking techniques (see e.g. Chapter 3 of this thesis). Hence, we can conclude that case studies have proved vital for the advancement of theory for μCRL.

The language μCRL$_t$ is still relatively young. One can argue, however, that its foundations have been around for over a decade, as μCRL$_t$ comes from a family of process algebras centred around ACP. The studies into real-time extensions of ACP have led to various distinct theories, which have been re-united only recently in [17]. However, these studies into the real-time extensions have been guided by the desire to understand the concept of time in computerised systems, rather than to investigate techniques for verification of real-time systems. The theory μCRL$_t$ aims to combine both the practical and the theoretical investigations.

At the moment of writing this, not much hands-on experience in using μCRL$_t$ for practical applications has been obtained. Several small-scale hybrid systems problems have been studied by Groote and van Wamel in [58], but apart from these, no attempt has been made to employ μCRL$_t$ on real-life systems. This is rather unfortunate, as such studies are much needed to investigate and classify the ease of its notation, its readability and the usability of its accompanying analytical techniques. The feedback, obtained by case studies can be used to guide investigations into techniques for μCRL$_t$, similar to the influence the case studies had on μCRL. Moreover, the position of μCRL$_t$ in relation to other real-time formalisms must be investigated, not only from a theoretical point of view (as is done in e.g. Chapter 6), but also from a practical point of view. Of course, apart from the benefits for μCRL$_t$, case studies lead to a better understanding of the systems that are being studied. In the end, this is hoped to have a positive influence on the development, stability and performance of future systems.

The example we study in this chapter is a typical hybrid system, consisting of a multi-purpose conveyor belt and a dedicated controller. We make an effort to describe the system as detailed as possible, and assess the current state of verification techniques for μCRL$_t$ by investigating the models of these systems.

The remainder of this chapter is organised as follows. The conveyor belt system, its characteristics and the control objectives are introduced in Section 7.1. Formal specifications of the conveyor belt and the controller are given in Section 7.2. Section 7.3 describes the process algebraic verification of the conveyor belt system specifications, whereas Section 7.4 contains the mathematical analysis of the system. Section 7.5 summarises the impressions and results obtained in the foregoing sections.

7.1 Conveyor Belt System

The system investigated in this chapter is, at first glance, a simple system, consisting of only a single basic component (a conveyor belt) that must be controlled to meet a number of objectives. In fact, the system has much in common with many of the (more elementary) buffer examples studied elsewhere in the literature. In this section, we introduce the conveyor belt and discuss the control objectives.

The conveyor belt is equipped with a rubber transportation belt, sensors and a motor for driving the belt. The information, obtained from sensors is communicated to its environment, which is, typically a controller. The motor receives its control via communication with its environment. The transportation belt is used for transporting trays and has a capacity limited to two trays. If more than two trays are on the belt, it breaks down. A schematic drawing of the belt, including its wirings to a controller, is found in Fig. 7.1. The belt has a fixed length $l > 0$. For the sake

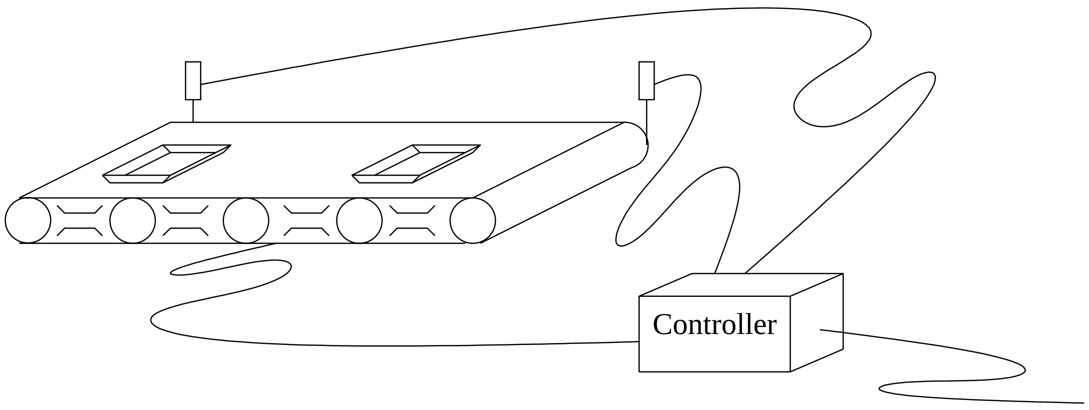


Figure 7.1: The Conveyor Belt

of argument, assume the maximal tray size is $s_t > 0$, and we have $l > 2s_t$. Below, we list the characteristics of the conveyor belt.

1. The sensors are triggered by the passing of trays,
2. The speed of the belt is determined by the force exerted by the motor,
3. The force exerted by the motor is determined by the torque, which is in the range of $[-T_{max}, T_{max}]$, where $T_{max} > 0$.
4. The belt cannot refuse trays,
5. Trays on the belt cannot overtake each-other.

The conveyor belt can be used in several application areas, and is a fairly generic component. It can, for instance, be used to temporarily store up to two trays, and then again deliver them in reverse order to the environment by running backward. Here, we study a less exotic use of the conveyor belt, viz. we use it to transport the trays from the front end of the belt to the back end of the belt. To this end, a controller of the conveyor belt must be specified. Below, we list the requirements the resulting system must satisfy.

1. At any time, at most two trays may reside on the belt,
2. A tray arriving at the belt may not collide into a tray already on the belt,
3. The velocity of the belt must be in the range of 0 to v_{max}, where $v_{max} > 0$ is the maximal speed,
4. Trays arriving at the front end of the belt are delivered to the back end of the belt,
5. If there are no trays on the belt, the belt eventually arrives to a full stop.

The first three requirements can be classified as safety requirements, the fourth requirement is a liveness requirement. The last requirement is an efficiency requirement, and directly influences the throughput and performance of the belt.

7.2 Formal Models

In the next section, we discuss two techniques for modelling the conveyor belt. Furthermore, we discuss the design of a controller for the belt satisfying the requirements, sketched informally in the previous section.

7.2.1 Conveyor Belt

The conveyor belt can be modelled using techniques, stemming from different backgrounds. The two models, explained in this section are typical models used in computer science and mechanical engineering.

We first present and discuss an automaton-like model of the conveyor belt system. The model, depicted in Fig. 7.2 shows various states and edges linking these states. The actions, decorating these edges represent the events that caused or are the result of taking the edge. Table 7.1 lists the events and their corresponding action. The discrete model nicely reflects the possible events the

Action	Event
a	Detection of a tray by the front-most sensor
d	Detection of a tray by the back-end sensor
$F(\mu)$	Adjusting the torque exerted on the motor to μ

Table 7.1: Actions corresponding to events

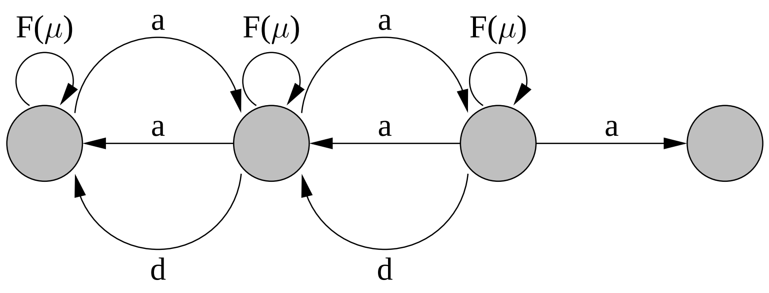


Figure 7.2: A purely discrete model of the conveyor belt

sensors on the belt register, and in which order. The sink state (i.e. the state without any outgoing edges) is used to denote that the belt has crashed. Any path with three successive arrivals of trays without deliveries of trays in between can lead to this state. The model also accounts for the running backward of the belt, which causes two successive a events to be registered, leaving the belt in a state in which there are again no (resp. one) trays on the belt. Figure 7.2 is very convenient for reachability analysis and can be used to show that, e.g. a given controller ensures no clashes of trays can occur. However, there is additional information that cannot be captured in this model, such as the (positive or negative) velocity and position of the trays on the belt. This extra information is useful for obtaining more optimal and reliable controllers.

The above mentioned shortcomings of a pure discrete model can, of course, be ignored. However, there is an abundance of mathematical theory that can readily model the missing information. For instance, the model, described by Eqn. (7.1) covers the dynamics of the tray (position and speed) and the belt (speed). The set of equations below is often referred to as a set of *Differential Algebraic Equations*, DAE for short.

$$\begin{aligned}
\dot{x}_0 &= bf \\
\dot{x}_1 &= \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} x_1 + \begin{pmatrix} 0 \\ b \end{pmatrix} f \\
\dot{x}_2 &= \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} x_2 + \begin{pmatrix} 0 \\ b \\ 0 \end{pmatrix} f
\end{aligned} \tag{7.1}$$

The value of $x_0(t)$ describes the velocity of the belt at time t when no tray is present on the belt. The first coordinate of $x_1(t)$ describes the position of the (only) tray on the belt, whereas the second coordinate describes the speed of the belt (and thus also of the tray). The tuple $x_2(t)$ denotes the position of the foremost tray at time t (first coordinate), the speed of the belt (second coordinate) and the position of the second tray (third coordinate). The function f represents the force that is exerted on the belt by the motor at any time, i.e. it is a function of the torque and time. Note that the constant b in the matrices accounts for physical forces, such as friction. The position of the trays on the belt corresponds to the position of a fixed reference point of the tray on the belt (see Fig. 7.3). This means that the sensors on the belt are triggered only by the passing of the reference point of a tray. Although this continuous model has accurate information concerning the position of trays on the belt and the speed of the belt, the link between the actual

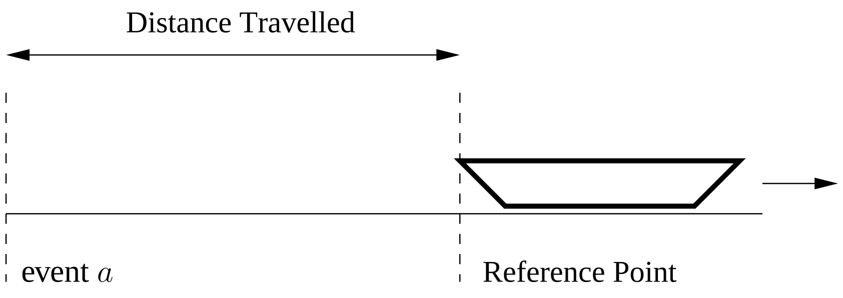


Figure 7.3: Fixed Reference Point of a Tray

number of trays on the belt and the equations is only specified informally.

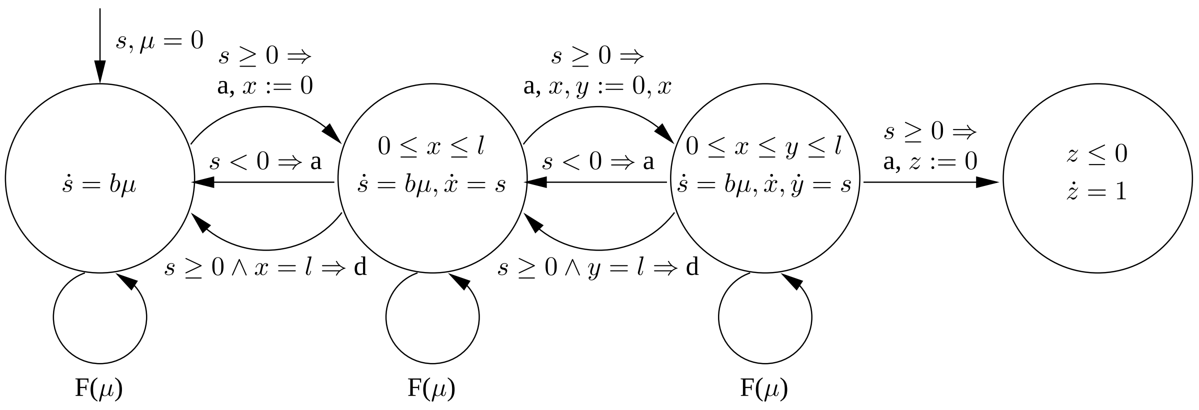

$s, \mu = 0$
$s \geq 0 \Rightarrow$ a, $x := 0$
$s < 0 \Rightarrow$ a
$s \geq 0 \wedge x = l \Rightarrow$ d
$\dot{s} = b\mu$
$0 \leq x \leq l$
$\dot{s} = b\mu, \dot{x} = s$
$s \geq 0 \Rightarrow$ a, $x, y := 0, x$
$s < 0 \Rightarrow$ a
$s \geq 0 \wedge y = l \Rightarrow$ d
$0 \leq x \leq y \leq l$
$\dot{s} = b\mu, \dot{x}, \dot{y} = s$
$s \geq 0 \Rightarrow$ a, $z := 0$
$z \leq 0$
$\dot{z} = 1$
F(μ)
F(μ)
F(μ)

Figure 7.4: Hybrid Model of the Conveyor Belt

Neither the discrete model, nor the continuous DAE model are capable of modelling the conveyor belt in all its detail. This is, of course, due to the hybrid nature of the conveyor belt, which is a mix of the continuous process of transportation with the discrete process of (binary) sensory information and control. Therefore, a more fitting model is the one depicted in Fig. 7.4. The hybrid model again has a sink state, in which we do not allow time to progress once we have arrived in this state. The model, depicted in Fig. 7.4 is inspired by the notation of hybrid automata, but allows for parameter passing actions (viz. the *F*(μ) action, setting the variable μ to a particular value). A formal μCRL$_t$ model is given in Table 7.3. Several abbreviations, used in the process definitions of Table 7.3 are defined in Table 7.2.

The μCRL$_t$ model combines the discrete transition model and the continuous DAE model, discussed prior. The events *a* and *d*, modelling the arrival and departure of trays on the belt are no longer non-deterministically determined, but can occur only under specific (disjoint) conditions. The resulting model therefore is a much more fitting abstraction of reality than both the purely discrete, or the continuous model.

The μCRL$_t$ model of Table 7.3 consists of three separate processes *B*$_i$, where the value i represents the current number of trays on the belt. The t parameter represents the time of execution of the latest action. The parameters x_s and x_{s_i} are used to register the last known states of the continuous processes. The function π_i, used in the guards for the actions is the mathematical projection function. The sink state, present in Figs. 7.2 and 7.4, is modelled by the process δꞏ**0**.

$$
\begin{array}{lcl}
\iota_0^d(x, x_s, \mu) & := & (\forall_{\zeta \in (0,d)} \dot{x}(\zeta) = b\mu) \\
 & & \wedge x(0) = x_s \\
\iota_1^d(x, x_s, \mu) & := & (\forall_{\zeta \in (0,d)} \dot{x}(\zeta) = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} x(\zeta) + \begin{pmatrix} 0 \\ b \end{pmatrix} \mu \wedge 0 \leq \pi_2(x(\zeta)) \leq l) \\
 & & \wedge x(0) = x_s \\
\iota_2^d(x, x_s, \mu) & := & (\forall_{\zeta \in (0,d)} \dot{x}(\zeta) = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} x(\zeta) + \begin{pmatrix} 0 \\ b \\ 0 \end{pmatrix} \mu \\
 & & \wedge 0 \leq \pi_1(x(\zeta)) \leq \pi_3(x(\zeta)) \leq l) \\
 & & \wedge x(0) = x_s
\end{array}
$$

Table 7.2: Abbreviations

The set of continuous functions that are differentiable almost everywhere is denoted by $\mathcal{F}$.

7.2.2 Controller

The μCRL$_t$ model of the conveyor belt does not yet satisfy any of the requirements specified in Section 7.1. For instance, the conveyor belt can break down by accepting a third tray while two trays already reside on the belt. Also, the belt can run either forward or backward, thereby having the possibility of delivering trays at both the front end and the back end of the belt.

Recall the criteria listed in section 7.1, to which the controller should adhere. Designing the controller is a process that is largely guided by these requirements. For instance, by carefully controlling the speed of the motor, we can ensure the belt runs in one direction only. On the basis of the continuous model, the position of the trays can be determined, providing a means for deciding on when it is safe to allow for another tray on the belt.

No Break-Down. By disallowing a third tray when there are already two trays on the belt, this requirement is easily fulfilled; hence, an obvious solution is to keep track of the number of trays currently on the belt.

No Collisions of Trays. The maximal size of a tray is $s_t < \frac{1}{2}l$. Hence, we should avoid allowing trays on the belt if there is a tray on the first half of the belt. Concretely, this means that if there is no tray on the belt, we can always accept trays, while if there is one tray on the belt, we accept trays only when this tray has travelled at least beyond the mid-point of the belt.

Delivery of Trays. Trays, arriving at the front end are transported to the back end of the belt if we can ensure the speed of the belt is positive. This requirement is therefore readily satisfied by ensuring the motor delivers maximal torque, whenever possible.

Speed Limits. The speed of the belt is controlled by means of setting the torque of the motor. Therefore, if maximal speed is reached, it suffices to continue running at this speed by setting the torque to zero. Whenever a stand-still is reached, we must ensure the torque is non-negative.

$$\begin{aligned}
&\textbf{proc } B_0(t{:}\mathbb{R}_{\geq \mathbf{0}}, x_s \in \mathbb{R}, \mu \in [-T_{max}, T_{max}]) = \\[1ex]
&\quad \textstyle\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \sum_{x_0:\mathcal{F}} \ \boldsymbol{a}\text{`}(t+d) \cdot B_1(t+d, (0, x_0(d)), \mu) \\
&\qquad \lhd\ \iota_0^d(x_0, x_s, \mu) \wedge x_0(d) \geq 0 \rhd \delta\text{`}\mathbf{0} \\
&+ \textstyle\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \sum_{x_0:\mathcal{F}} \sum_{\mu' \in [-T_{max}, T_{max}]} \ \boldsymbol{F(\mu')}\text{`}(t+d) \cdot B_0(t+d, x_0(d), \mu') \\
&\qquad \lhd\ \iota_0^d(x, x_s, \mu) \rhd \delta\text{`}\mathbf{0}
\end{aligned}$$

$$\begin{aligned}
&\textbf{proc } B_1(t{:}\mathbb{R}_{\geq \mathbf{0}}, (x_{s_0}, x_{s_1}) \in \mathbb{R}^2, \mu \in [-T_{max}, T_{max}]) = \\[1ex]
&\quad \textstyle\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \sum_{x_1:\mathcal{F}} \ \boldsymbol{a}\text{`}(t+d) \cdot B_2(t+d, (0, x_1(d)), \mu) \\
&\qquad \lhd\ \iota_1^d(x_1, (x_{s_0}, x_{s_1}), \mu) \wedge \pi_1(x_1(d)) \geq 0 \rhd \delta\text{`}\mathbf{0} \\
&+ \textstyle\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \sum_{x_1:\mathcal{F}} \ \boldsymbol{a}\text{`}(t+d) \cdot B_0(t+d, \pi_1(x_1(d)), \mu) \\
&\qquad \lhd\ \iota_1^d(x_1, (x_{s_0}, x_{s_1}), \mu) \wedge \pi_1(x_1(d)) < 0 \wedge \pi_2(x_1(d)) = 0 \rhd \delta\text{`}\mathbf{0} \\
&+ \textstyle\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \sum_{x_1:\mathcal{F}} \ \boldsymbol{d}\text{`}(t+d) \cdot B_0(t+d, \pi_1(x_1(d)), \mu) \\
&\qquad \lhd\ \iota_1^d(x_1, (x_{s_0}, x_{s_1}), \mu) \wedge \pi_2(x_1(d)) = l \wedge \pi_1(x_1(d)) \geq 0 \rhd \delta\text{`}\mathbf{0} \\
&+ \textstyle\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \sum_{x_1:\mathcal{F}} \sum_{\mu' \in [-T_{max}, T_{max}]} \ \boldsymbol{F(\mu')}\text{`}(t+d) \cdot B_1(t+d, x_1(d), \mu') \\
&\qquad \lhd\ \iota_1^d(x_1, (x_{s_0}, x_{s_1}), \mu) \wedge 0 \leq \pi_2(x_1(d)) \leq l \rhd \delta\text{`}\mathbf{0}
\end{aligned}$$

$$\begin{aligned}
&\textbf{proc } B_2(t{:}\mathbb{R}_{\geq \mathbf{0}}, (x_{s_0}, x_{s_1}, x_{s_2}) \in \mathbb{R}^3, \mu \in [-T_{max}, T_{max}]) = \\[1ex]
&\quad \textstyle\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \sum_{x_2:\mathcal{F}} \ \boldsymbol{a}\text{`}(t+d) \cdot \delta\text{`}\mathbf{0} \\
&\qquad \lhd\ \iota_2^d(x_2, (x_{s_0}, x_{s_1}, x_{s_2}), \mu) \wedge \pi_2(x_2(d)) \geq 0 \rhd \delta\text{`}\mathbf{0} \\
&+ \textstyle\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \sum_{x_2:\mathcal{F}} \ \boldsymbol{a}\text{`}(t+d) \cdot B_1(t+d, (\pi_1(x_2(d)), \pi_2(x_2(d))), \mu) \\
&\qquad \lhd\ \iota_2^d(x_2, (x_{s_0}, x_{s_1}, x_{s_2}), \mu) \wedge \pi_2(x_2(d)) < 0 \wedge \pi_1(x_2(d)) = 0 \rhd \delta\text{`}\mathbf{0} \\
&+ \textstyle\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \sum_{x_2:\mathcal{F}} \ \boldsymbol{d}\text{`}(t+d) \cdot B_1(t+d, (\pi_2(x_1(d)), \pi_3(x_1(d)), \mu) \\
&\qquad \lhd\ \iota_2^d(x_2, (x_{s_0}, x_{s_1}, x_{s_2}), \mu) \wedge \pi_3(x_2(d)) = l \wedge \pi_2(x_2(d)) \geq 0 \rhd \delta\text{`}\mathbf{0} \\
&+ \textstyle\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \sum_{x_2:\mathcal{F}} \sum_{\mu' \in [-T_{max}, T_{max}]} \ \boldsymbol{F(\mu')}\text{`}(t+d) \cdot B_2(t+d, x_2(d), \mu') \\
&\qquad \lhd\ \iota_2^d(x_2, (x_{s_0}, x_{s_1}, x_{s_2}), \mu) \wedge 0 \leq \pi_1(x_2(d)) \leq \pi_3(x_2(d)) \leq l \rhd \delta\text{`}\mathbf{0}
\end{aligned}$$

Table 7.3: Hybrid Model of Conveyor Belt in μCRL$_t$

Action	Event
$\overline{a}$	Receiving sensory information from the front-most sensor
$\overline{d}$	Receiving sensory information from the back-end sensor
$\overline{F}(\mu)$	Updating the torque exerted by the motor to μ
$\overline{c}$	Signalling the environment it can deliver another tray

Table 7.4: Actions corresponding to events

Preserving Energy. If there is no tray on the belt while the speed of the belt is still greater than zero, we require a deceleration of the belt by setting the torque to minimal. We choose to set in the deceleration after a delay of $\mathfrak{d}$ time-units have passed. This allows for a greater flexibility in controlling the belt optimally in a variety of environments.

In order for the controller to function properly, it must have different interfaces with its environment. It must be able to receive sensory information from the belt, and send control signals to the motor, driving the belt. Moreover, it needs an interface with the outside world to signal when it is safe to allow another tray on the belt. The actions, corresponding to these events are listed in Table 7.4. The control criteria and their resulting design decision, together with the actions modelling events culminate into a solution for the controller, shown in Table 7.6. We have used several abbreviations, which are listed in Table 7.5.

$$
\begin{array}{lll}
\theta_0^d(x, x_s, \mu) & := & \forall_{\zeta \in (0,d)}(\dot{x}(\zeta) = b\mu \wedge 0 \leq x(\zeta) \leq v_{max}) \\
 & & \wedge x(0) = x_s \\
\theta_1^d(x, x_s, \mu) & := & \forall_{\zeta \in (0,d)}(\dot{x}(\zeta) = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} x(\zeta) + \begin{pmatrix} 0 \\ b \end{pmatrix} \mu \\
 & & \wedge 0 \leq \pi_1(x(\zeta)) \leq v_{max} \wedge \pi_2(x(\zeta)) \leq \frac{1}{2}l) \wedge x(0) = x_s
\end{array}
$$

Table 7.5: Abbreviations

7.3 Process Algebraic Verification of the Belt System

The μCRL$_t$ model for the conveyor belt and its controller, provided in Tables 7.3 and 7.6 are fairly complex. This is mostly due to the complexity of the conditions, acting as guards for the timed executions of actions. The analysis in the following section shows the complexity of most conditions can be considerably reduced. Moreover, by rewriting the specifications into a *Timed Linear Process Equations* format (see Chapter 2), further analysis of the composition of the two processes is supported. An explanation of the linearisation process is given in Section 7.3.2. We first focus on the simplifications of the conditions.

$$\textbf{proc } C = \overline{c}\text{‹}\mathbf{0} \cdot C_0(0,0,0)$$

$$\begin{aligned}
&\textbf{proc } C_0(t{:}\mathbb{R}_{\geq\mathbf{0}}, x_s \in \mathbb{R}, \mu \in [-T_{max}, T_{max}]) = \\
&\quad \textstyle\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \sum_{x_0:\mathcal{F}} \overline{a}\text{‹}(t+d) \cdot C^{-}_{1,-}(t+d, (x_0(d), 0), \mu) \\
&\qquad \lhd\, \bar{\theta}^d_0(x_0, x_s, \mu) \wedge 0 \leq x_0(d) \leq v_{max} \rhd \delta\text{‹}\mathbf{0} \\
&+ \textstyle\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \sum_{x_0:\mathcal{F}} \overline{F}(0)\text{‹}(t+d) \cdot C_0(t+d, 0, 0) \\
&\qquad \lhd\, \bar{\theta}^d_0(x_0, x_s, \mu) \wedge 0 \leq x_0(d) \leq v_{max} \wedge \mu \neq 0 \rhd \delta\text{‹}\mathbf{0}
\end{aligned}$$

$$\begin{aligned}
&\textbf{proc } C^{+}_0(t, \Delta{:}\mathbb{R}_{\geq\mathbf{0}}, x_s \in \mathbb{R}, \mu \in [-T_{max}, T_{max}]) = \\
&\quad \textstyle\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \sum_{x_0:\mathcal{F}} \overline{a}\text{‹}(t+d) \cdot C^{+}_{1,-}(t+d, (x_0(d), 0), \mu) \\
&\qquad \lhd\, \bar{\theta}^d_0(x_0, x_s, \mu) \wedge 0 \leq x_0(d) \leq v_{max} \wedge d \leq \Delta \rhd \delta\text{‹}\mathbf{0} \\
&+ \textstyle\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \sum_{x_0:\mathcal{F}} \overline{F}(0)\text{‹}(t+d) \cdot C^{+}_0(t+d, \Delta - d, v_{max}, 0) \\
&\qquad \lhd\, \bar{\theta}^d_0(x_0, x_s, \mu) \wedge x_0(d) = v_{max} \wedge d \leq \Delta \wedge \mu \neq 0 \rhd \delta\text{‹}\mathbf{0} \\
&+ \textstyle\sum_{x_0:\mathcal{F}} \overline{F}(-T_{max})\text{‹}(t+\Delta) \cdot C_0(t+\Delta, x_0(\Delta), -T_{max}) \\
&\qquad \lhd\, \theta^{\Delta}_0(x_0, x_s, \mu) \wedge 0 \leq x_0(\Delta) \leq v_{max} \rhd \delta\text{‹}\mathbf{0}
\end{aligned}$$

$$\begin{aligned}
&\textbf{proc } C^{-}_{1,-}(t{:}\mathbb{R}_{\geq\mathbf{0}}, (x_{s_0}, x_{s_1}) \in \mathbb{R}^2, \mu \in [-T_{max}, T_{max}]) = \\
&\quad \overline{F}(0)\text{‹}t \cdot C^{+}_{1,-}(t, (x_{s_0}, x_{s_1}), 0) \lhd x_{s_0} = v_{max} \rhd \overline{F}(T_{max})\text{‹}t \cdot C^{+}_{1,-}(t, (x_{s_0}, x_{s_1}), T_{max})
\end{aligned}$$

$$\begin{aligned}
&\textbf{proc } C^{+}_{1,-}(t{:}\mathbb{R}_{\geq\mathbf{0}}, (x_{s_0}, x_{s_1}) \in \mathbb{R}^2, \mu \in [-T_{max}, T_{max}]) = \\
&\quad \textstyle\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \sum_{x_1:\mathcal{F}} \overline{F}(0)\text{‹}(t+d) \cdot C^{+}_{1,-}(t+d, x_1(d), 0) \\
&\qquad \lhd\, \bar{\theta}^d_1(x_1, (x_{s_0}, x_{s_1}), \mu) \wedge \pi_1(x_1(d)) = v_{max} \wedge \pi_2(x_1(d)) \leq \tfrac{1}{2}l \wedge \mu \neq 0 \rhd \delta\text{‹}\mathbf{0} \\
&+ \textstyle\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \sum_{x_1:\mathcal{F}} \overline{c}\text{‹}(t+d) \cdot C_{1,+}(t+d, x_1(d), \mu) \\
&\qquad \lhd\, \bar{\theta}^d_1(x_1, (x_{s_0}, x_{s_1}), \mu) \wedge 0 \leq \pi_1(x_1(d)) \leq v_{max} \wedge \pi_2(x_1(\zeta)) = \tfrac{1}{2}l \rhd \delta\text{‹}\mathbf{0}
\end{aligned}$$

$$\begin{aligned}
&\textbf{proc } C_{1,+}(t{:}\mathbb{R}_{\geq\mathbf{0}}, x_s \in \mathbb{R}, \mu \in [-T_{max}, T_{max}]) = \\
&\quad \textstyle\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \sum_{x_0:\mathcal{F}} \overline{a}\text{‹}(t+d) \cdot C_2(u, (x_0(d), 0), \mu) \\
&\qquad \lhd\, \bar{\theta}^d_0(x_0, x_s, \mu) \wedge x_0(d) \leq v_{max} \rhd \delta\text{‹}\mathbf{0} \\
&+ \textstyle\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \sum_{x_0:\mathcal{F}} \overline{d}\text{‹}(t+d) \cdot C^{+}_0(t+d, \mathfrak{d}, x_0(d), \mu) \\
&\qquad \lhd\, \bar{\theta}^d_0(x_0, x_s, \mu) \wedge x_0(d) \leq v_{max} \rhd \delta\text{‹}\mathbf{0} \\
&+ \textstyle\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \sum_{x_0:\mathcal{F}} \overline{F}(0)\text{‹}(t+d) \cdot C_{1,+}(t+d, x_0(d), 0) \\
&\qquad \lhd\, \bar{\theta}^d_0(x_0, x_s, \mu) \wedge x_0(d) = v_{max} \wedge \mu \neq 0 \rhd \delta\text{‹}\mathbf{0}
\end{aligned}$$

$$\begin{aligned}
&\textbf{proc } C_2(t{:}\mathbb{R}_{\geq\mathbf{0}}, (x_{s_0}, x_{s_1}) \in \mathbb{R}^2, \mu \in [-T_{max}, T_{max}]) = \\
&+ \textstyle\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \sum_{x_1:\mathcal{F}} \overline{d}\text{‹}(t+d) \cdot C^{+}_{1,-}(t+d, x_1(d), \mu) \\
&\qquad \lhd\, \bar{\theta}^d_1(x_1, (x_{s_0}, x_{s_1}), \mu) \wedge \pi_1(x_1(d)) \leq v_{max} \rhd \delta\text{‹}\mathbf{0} \\
&+ \textstyle\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \sum_{x_1:\mathcal{F}} \overline{F}(0)\text{‹}(t+d) \cdot C_{1,+}(t+d, x_1(d), 0) \\
&\qquad \lhd\, \bar{\theta}^d_1(x_1, (x_{s_0}, x_{s_1}), \mu) \wedge \pi_1(x_1(d)) = v_{max} \wedge \mu \neq 0 \rhd \delta\text{‹}\mathbf{0}
\end{aligned}$$

Table 7.6: Controller

7.3.1 Reducing Complex Conditions

The conditions, guarding the actions, all have differential equations, stemming from the continuous model, in common. These differential equations are relatively simple; the solutions to these equations can be calculated exactly and do not need to be approximated, as is shown in the Eqns. (7.2) through (7.4) below.

$$\frac{dx_0}{dt} = b\mu \Leftrightarrow \exists_{C\in\mathbb{R}} x_0(t) = b\mu t + C \tag{7.2}$$

$$\begin{pmatrix} \frac{dx_1}{dt} \\ \frac{dx_1'}{dt} \end{pmatrix} = \begin{pmatrix} x_1' \\ b\mu \end{pmatrix} \Leftrightarrow \exists_{C,C'\in\mathbb{R}} \begin{cases} x_1(t) = \frac{1}{2}b\mu t^2 + C't + C \wedge \\ x_1'(t) = b\mu t + C' \end{cases} \tag{7.3}$$

$$\begin{pmatrix} \frac{dx_2}{dt} \\ \frac{dx_2'}{dt} \\ \frac{dx_2''}{dt} \end{pmatrix} = \begin{pmatrix} x_2' \\ b\mu \\ x_2' \end{pmatrix} \Leftrightarrow \exists_{C,C',C''\in\mathbb{R}} \begin{cases} x_2(t) = \frac{1}{2}b\mu t^2 + C't + C \wedge \\ x_2'(t) = b\mu t + C' \wedge \\ x_2''(t) = \frac{1}{2}b\mu t^2 + C't + C'' \end{cases} \tag{7.4}$$

The above functions, in fact, are the well-known laws of physics, identified by Newton several centuries earlier. The laws relate acceleration, speed and position of an object to the passage of time.

As we have additional knowledge on the (continuous) state after executing an action, we are in a position where we can determine the constants C, C' and C'' (i.e. we have so-named *initial values*). Thus, rather than considering all functions satisfying the differential equations and their side conditions, it suffices to consider fixed, *a priori* known functions. We therefore have enough information to eliminate all of the $\sum$ operators, quantifying over the function space $\mathcal{F}$.

7.3.2 Linearisation of Formal Models

The complexity reductions, discussed in the previous section, give rise to more understandable process descriptions. There is, however, no way of telling whether the controller indeed guarantees the belt behaves functionally correct. For instance, it is imaginable that the controller allows a third tray to arrive at the belt, thereby physically “breaking” the belt. A thorough investigation of the controller therefore requires an analysis of the parallel composition of the controller and the belt.

In a setting without time, the technique of *linearisation* is used to simplify the somewhat laborious task of determining the exact behaviour of two processes in parallel. For processes in linear form, also *Linear Process Equations*, a technique for eliminating parallelism exists. In a setting with time, similar techniques have been developed (see e.g. [57]). Hence, an almost standard procedure is to rewrite the processes representing the controller and the belt into linear form. The resulting models are listed in Tables 7.7 and 7.8.

Several remarks are in order. The processes in Table 7.7 and 7.8 have a slightly altered domain (by the domain of a process we mean the list of parameters of the process), compared to their original definition. This is due to the linearisation. For the belt, we have used the following encoding. The value for σ represents process B_σ, s represents the velocity of the belt, x represents the position of the trailing tray (which can be the only tray on the belt), and y represents the position of the leading tray (in case there are two trays on the belt). For the controller, we have introduced a parameter ς to represent the ς^{th} process in the order of appearance of Table 7.6

(starting from zero). Hence, $\varsigma = 2$ represents process C_0^+. Parameter s in process C represents the velocity of the belt according to the controller, x represents the position of the trailing tray on the belt and ϱ represents the torque exerted by the motor.

7.3.3 Encapsulating and Expanding

The controller, defined in the previous section, is a rather complex system. So far, we have only argued informally that, by design, it ensures all requirements posed on the conveyor belt are met. To put these informal arguments to the test, we investigate the behaviour of the conveyor belt as it is controlled by the controller, using rigorous formal techniques, developed for μCRL$_t$.

Communications

Recall that the conveyor belt can interact with its environment via actions, modelling sensory information and inputs for motor settings. Likewise, the controller can receive information from, and send valuable information to its environment. Table 7.9 lists the intended communications for the belt and the controller. Assume the communications function γ satisfies Table 7.9. The intention is to study the behaviour of the belt under the control of our controller. Therefore, we assume neither belt, nor controller can autonomously execute actions that should synchronise. Thus, the system *S* we wish to study is specified by Eqn. (7.5).

$$\begin{array}{ll} & S(t,u,\Delta{:}\mathbb{R}_{\geq \mathbf{0}},\sigma \in \{0,1,2\},\varsigma \in \{0,..,6\},s_b,x_b,y_b,s_c,x_c{:}\mathbb{R},\mu,\varrho \in [-T_{max},T_{max}]) \\ = & \\ & \partial_{\{a,d,F,\overline{a},\overline{d},\overline{F}\}}(B(t,\sigma,s_b,x_b,y_b,\mu) \| C(u,\Delta,s_c,x_c,\varrho)) \end{array} \tag{7.5}$$

We write H for the set $\{a,d,F,\overline{a},\overline{d},\overline{F}\}$, and refer to it as the *encapsulation* set. The process $S(\overline{0})$, representing the interactions between the belt and controller is studied in the next section.

Encapsulating and Expanding

Unfortunately, we have no tools at our disposal for analysing the behaviour of the composition of two systems without first eliminating all occurrences of parallel composition. This process of elimination is here called *encapsulation and expansion*, and is so-named after the encapsulation and expansion theorems of μCRL$_t$ (see e.g. [57]). Encapsulation and expansion of timed parallel processes is a cumbersome task with a rather high complexity. When the individual processes are well understood, educated guesses can reveal several desired invariants of the system. These invariants are often shown to be correct in retrospect of the expansion and encapsulation of the parallelism, but they can also be used to characterise all reachable states during expansion and encapsulation. In case a non-invariant property is used to reduce the complexity of encapsulation and expansion, this is always detected during this process.

To sketch an idea of the complexity of the encapsulation and expansion, we can do the following calculations. The linearised process *B* consists of a total of 7 alternative *a* actions, 4 alternative *d* actions and 3 alternative *F* actions, whereas the linearised process *C* consists of 3 alternative $\overline{c}$ actions, 3 alternative $\overline{a}$ actions, 2 alternative $\overline{d}$ actions and 8 alternative $\overline{F}$ actions. A straightforward application of encapsulation and expansion therefore yields a process consisting of 21 alternative a actions, 8 alternative d actions, 24 alternative F actions, 42 alternative $\overline{c}$ actions

proc $B(t{:}\mathbb{R}_{\geq\mathbf{0}}, \sigma \in \{0,1,2\}, s,x,y \in \mathbb{R}, \mu \in [-T_{max}, T_{max}]) =$

(B1) $\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} a{}^\backprime(t+d) \cdot B(t+d, 1, b\mu d + s, 0, y, \mu)$
$\quad \lhd \sigma = 0 \wedge s + b\mu d \geq 0 \rhd \delta{}^\backprime\mathbf{0} +$
(B2) $\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \sum_{\mu' \in [-T_{max}, T_{max}]} F(\mu'){}^\backprime(t+d) \cdot B(t+d, 0, b\mu d + s, x, y, \mu')$
$\quad \lhd \sigma = 0 \rhd \delta{}^\backprime\mathbf{0} +$
(B3) $\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} a{}^\backprime(t+d) \cdot B(t+d, 2, b\mu d + s, 0, \frac{1}{2}b\mu d^2 + sd + x, \mu)$
$\quad \lhd \sigma = 1 \wedge s + b\mu d \geq 0 \wedge 0 \leq \frac{1}{2}b\mu d^2 + sd + x \leq l \rhd \delta{}^\backprime\mathbf{0} +$
(B4) $a{}^\backprime(t + \frac{-s-\sqrt{s^2-2b\mu x}}{b\mu}) \cdot B(t + \frac{-s-\sqrt{s^2-2b\mu x}}{b\mu}, 0, -\sqrt{s^2-2b\mu x}, x, y, \mu)$
$\quad \lhd \sigma = 1 \wedge b\mu \neq 0 \wedge \sqrt{s^2-2b\mu x} > 0 \rhd \delta{}^\backprime\mathbf{0} +$
(B5) $a{}^\backprime(t + \frac{-x}{s}) \cdot B(t + \frac{-x}{s}, 0, s, x, y, \mu)$
$\quad \lhd \sigma = 1 \wedge b\mu = 0 \wedge s < 0 \rhd \delta{}^\backprime\mathbf{0} +$
(B6) $d{}^\backprime(t + \frac{-s+\sqrt{s^2-2b\mu(x-l)}}{b\mu}) \cdot B(t + \frac{-s+\sqrt{s^2-2b\mu(x-l)}}{b\mu}, 0, \sqrt{s^2-2b\mu(x-l)}, x, y, \mu)$
$\quad \lhd \sigma = 1 \wedge b\mu \neq 0 \wedge \sqrt{s^2-2b\mu(x-l)} \geq 0 \rhd \delta{}^\backprime\mathbf{0} +$
(B7) $d{}^\backprime(t + \frac{l-x}{s}) \cdot B(t + \frac{l-x}{s}, 0, s, x, y, \mu)$
$\quad \lhd \sigma = 1 \wedge b\mu = 0 \wedge s \geq 0 \rhd \delta{}^\backprime\mathbf{0} +$
(B8) $\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \sum_{\mu' \in [-T_{max}, T_{max}]} F(\mu'){}^\backprime(t+d) \cdot B(t+d, 1, s + b\mu d, \frac{1}{2}b\mu d^2 + sd + x, y, \mu')$
$\quad \lhd \sigma = 1 \wedge 0 \leq \frac{1}{2}b\mu d^2 + sd + x \leq l \rhd \delta{}^\backprime\mathbf{0} +$
(B9) $\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} a{}^\backprime(t+d) \cdot \delta{}^\backprime\mathbf{0}$
$\quad \lhd \sigma = 2 \wedge s + b\mu d \geq 0 \wedge 0 \leq \frac{1}{2}b\mu d^2 + sd + x \leq \frac{1}{2}b\mu d^2 + sd + y \leq l \rhd \delta{}^\backprime\mathbf{0} +$
(B10) $a{}^\backprime(t + \frac{-s-\sqrt{s^2-2b\mu x}}{b\mu}) \cdot B(t + \frac{-s-\sqrt{s^2-2b\mu x}}{b\mu}, 1, -\sqrt{s^2-2b\mu x}, y-x, y, \mu)$
$\quad \lhd \sigma = 2 \wedge b\mu \neq 0 \wedge \sqrt{s^2-2b\mu x} < 0 \rhd \delta{}^\backprime\mathbf{0} +$
(B11) $a{}^\backprime(t + \frac{x}{s}) \cdot B(t + \frac{x}{s}, 1, s, y-x, y, \mu)$
$\quad \lhd \sigma = 2 \wedge b\mu = 0 \wedge s < 0 \rhd \delta{}^\backprime\mathbf{0} +$
(B12) $d{}^\backprime(t + \frac{-s+\sqrt{s^2-2b\mu(y-l)}}{b\mu}) \cdot B(t + \frac{-s+\sqrt{s^2-2b\mu(y-l)}}{b\mu}, 1,$
$\quad \sqrt{s^2-2b\mu(y-l)}, x + (l-y), y, \mu)$
$\quad \lhd \sigma = 2 \wedge b\mu \neq 0 \wedge \sqrt{s^2-2b\mu(y-l)} \geq 0 \rhd \delta{}^\backprime\mathbf{0} +$
(B13) $d{}^\backprime(t + \frac{l-y}{s}) \cdot B(t + \frac{l-y}{s}, 1, s, x + (l-y), y, \mu)$
$\quad \lhd \sigma = 2 \wedge b\mu = 0 \wedge s \geq 0 \rhd \delta{}^\backprime\mathbf{0} +$
(B14) $\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \sum_{\mu' \in [-T_{max}, T_{max}]} F(\mu'){}^\backprime(t+d) \cdot$
$\quad B(t+d, 2, s + b\mu d, \frac{1}{2}b\mu d^2 + sd + x, \frac{1}{2}b\mu d^2 + sd + y, \mu')$
$\quad \lhd \sigma = 2 \wedge 0 \leq \frac{1}{2}b\mu d^2 + sd + x \leq \frac{1}{2}b\mu d^2 + sd + y \leq l \rhd \delta{}^\backprime\mathbf{0}$

Table 7.7: Linearised and Reduced μCRL$_t$-Model of the Conveyor Belt

proc $C(t, \Delta{:}\mathbb{R}_{\geq\mathbf{0}}, \sigma \in \{0, ..., 6\}, s, x \in \mathbb{R}, \mu \in [-T_{max}, T_{max}]) =$

(C1) $\overline{c}{\scriptstyle\mathsf{c}}0 \cdot C(0, 0, 1, 0, 0, 0)$
$\quad \lhd \sigma = 0 \rhd \delta{\scriptstyle\mathsf{c}}\mathbf{0} +$
(C2) $\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \overline{a}{\scriptstyle\mathsf{c}}(t + d) \cdot C(t + d, \Delta, 3, s + b\mu d, 0, \mu)$
$\quad \lhd \sigma = 1 \wedge s \geq 0 \wedge 0 \leq s + b\mu d \leq v_{max} \rhd \delta{\scriptstyle\mathsf{c}}\mathbf{0} +$
(C3) $\overline{F}(0){\scriptstyle\mathsf{c}}(t + \frac{-s}{b\mu}) \cdot C(t + \frac{-s}{b\mu}, \Delta, 1, 0, x, 0)$
$\quad \lhd \sigma = 1 \wedge b\mu \neq 0 \rhd \delta{\scriptstyle\mathsf{c}}\mathbf{0} +$
(C4) $\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \overline{a}{\scriptstyle\mathsf{c}}(t + d) \cdot C(t + d, \Delta, 4, s + b\mu d, 0, \mu)$
$\quad \lhd \sigma = 2 \wedge 0 \leq s + b\mu d \leq v_{max} \wedge d \leq \Delta \rhd \delta{\scriptstyle\mathsf{c}}\mathbf{0} +$
(C5) $\overline{F}(0){\scriptstyle\mathsf{c}}(t + \frac{v_{max} - s}{b\mu}) \cdot C(t + \frac{v_{max} - s}{b\mu}, \Delta - \frac{v_{max} - s}{b\mu}, 2, v_{max}, x, 0)$
$\quad \lhd \sigma = 2 \wedge b\mu \neq 0 \wedge \frac{v_{max} - s}{b\mu} \leq \Delta \rhd \delta{\scriptstyle\mathsf{c}}\mathbf{0} +$
(C6) $\overline{F}(-T_{max}){\scriptstyle\mathsf{c}}(t + \Delta) \cdot C(t + \Delta, \Delta, 1, s + b\mu\Delta, x, -T_{max})$
$\quad \lhd \sigma = 2 \wedge 0 \leq s + b\mu\Delta \leq v_{max} \rhd \delta{\scriptstyle\mathsf{c}}\mathbf{0} +$
(C7) $\overline{F}(0){\scriptstyle\mathsf{c}}t \cdot C(t, \Delta, 4, s, x, 0)$
$\quad \lhd \sigma = 3 \wedge s = v_{max} \rhd \delta{\scriptstyle\mathsf{c}}\mathbf{0} +$
(C8) $\overline{F}(T_{max}){\scriptstyle\mathsf{c}}t \cdot C(t, \Delta, 4, s, x, T_{max})$
$\quad \lhd \sigma = 3 \wedge s \neq v_{max} \rhd \delta{\scriptstyle\mathsf{c}}\mathbf{0} +$
(C9) $\overline{F}(0){\scriptstyle\mathsf{c}}(t + \frac{v_{max} - s}{b\mu}) \cdot C(t + \frac{c_{max} - s}{b\mu}, \Delta, 4, v_{max}, \frac{1}{2}b\mu(\frac{v_{max} - s}{b\mu})^2 + s\frac{v_{max} - s}{b\mu} + x, 0)$
$\quad \lhd \sigma = 4 \wedge \frac{1}{2}b\mu(\frac{v_{max} - s}{b\mu})^2 + s\frac{v_{max} - s}{b\mu} + x \leq \frac{1}{2}l \wedge b\mu \neq 0 \rhd \delta{\scriptstyle\mathsf{c}}\mathbf{0} +$
(C10) $\overline{c}{\scriptstyle\mathsf{c}}(t + \frac{-s + \sqrt{s^2 - 2b\mu(x - \frac{1}{2}l)}}{b\mu}) \cdot C(t + \frac{-s + \sqrt{s^2 - b\mu(2x - l)}}{b\mu}, \Delta, 5, \sqrt{s^2 - b\mu(2x - l)}, \frac{1}{2}l, \mu)$
$\quad \lhd \sigma = 4 \wedge \sqrt{s^2 - b\mu(2x - l)} \leq v_{max} \wedge b\mu \neq 0 \rhd \delta{\scriptstyle\mathsf{c}}\mathbf{0} +$
(C11) $\overline{c}{\scriptstyle\mathsf{c}}(t + \frac{\frac{1}{2}l - x}{s}) \cdot C(t + \frac{\frac{1}{2}l - x}{s}, \Delta, 5, v_{max}, \frac{1}{2}l, \mu)$
$\quad \lhd \sigma = 4 \wedge b\mu = 0 \rhd \delta{\scriptstyle\mathsf{c}}\mathbf{0} +$
(C12) $\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \overline{a}{\scriptstyle\mathsf{c}}(t + d) \cdot C(t + d, \Delta, 6, s + b\mu d, 0, \mu)$
$\quad \lhd \sigma = 5 \wedge 0 \leq s + b\mu d \leq v_{max} \rhd \delta{\scriptstyle\mathsf{c}}\mathbf{0} +$
(C13) $\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \overline{\mathsf{d}}{\scriptstyle\mathsf{c}}(t + d) \cdot C(t + d, \mathfrak{d}, 2, s + b\mu d, x, \mu)$
$\quad \lhd \sigma = 5 \wedge 0 \leq s + b\mu d \leq v_{max} \rhd \delta{\scriptstyle\mathsf{c}}\mathbf{0} +$
(C14) $\overline{F}(0){\scriptstyle\mathsf{c}}(t + \frac{v_{max} - s}{b\mu}) \cdot C(t + \frac{v_{max} - s}{b\mu}, \Delta, 5, v_{max}, \frac{1}{2}b\mu(\frac{v_{max} - s}{b\mu})^2 + s(\frac{v_{max} - s}{b\mu} + x, 0)$
$\quad \lhd \sigma = 5 \wedge b\mu \neq 0 \rhd \delta{\scriptstyle\mathsf{c}}\mathbf{0} +$
(C15) $\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \overline{\mathsf{d}}{\scriptstyle\mathsf{c}}(t + d) \cdot C(t + d, \Delta, 4, s + b\mu d, \frac{1}{2}b\mu d^2 + sd + x, \mu)$
$\quad \lhd \sigma = 6 \wedge 0 \leq s + b\mu d \leq v_{max} \rhd \delta{\scriptstyle\mathsf{c}}\mathbf{0} +$
(C16) $\overline{F}(0){\scriptstyle\mathsf{c}}(t + \frac{v_{max} - s}{b\mu}) \cdot C(t + \frac{v_{max} - s}{b\mu}, \Delta, 6, v_{max}, \frac{1}{2}b\mu(\frac{v_{max} - s}{b\mu})^2 + s\frac{v_{max} - s}{b\mu} + x, 0)$
$\quad \lhd \sigma = 6 \wedge b\mu \neq 0 \rhd \delta{\scriptstyle\mathsf{c}}\mathbf{0}$

Table 7.8: Linearised and Reduced μCRL$_t$-Model of the Controller

Belt	Controller	Communication
a	$\overline{a}$	$\mathbf{a}$
d	$\overline{d}$	$\mathbf{d}$
$F(\mu)$	$\overline{F}(\mu)$	$\mathbf{F}(\mu)$

Table 7.9: Intended Communications between Belt and Controller

1.	$\sigma = 0 \Leftrightarrow \varsigma \in \{0,1,2\}$,
2.	$\sigma = 1 \Leftrightarrow \varsigma \in \{3,4,5\}$,
3.	$\sigma = 2 \Leftrightarrow \varsigma = 6$,
4.	$\mu = \varrho$,
5.	$u \geq t$,
6.	$\varsigma \neq 5 \Rightarrow u = t$,
7.	$0 \leq s_b \leq s_c \leq v_{max}$,
8.	$s_c = s_b + b\mu(u-t)$,
9.	$0 \leq x_b \leq l$,
10.	$x_c = x_b + \frac{1}{2}b\mu(u-t)^2 + s_b(u-t)$,
11.	$\varsigma \leq 4 \Rightarrow 0 \leq x_c \leq \frac{1}{2}l$,
12.	$\varsigma = 6 \Rightarrow y_b - x_b \geq \frac{1}{2}l$,
13.	$\varsigma \notin \{1,3\} \Rightarrow b\mu \geq 0$,
14.	$\varsigma \notin \{1,3\} \wedge b\mu = 0 \Rightarrow s_c = v_{max}$,
15.	$\varsigma \in \{1,3\} \Rightarrow b\mu \leq 0$,
16.	$\Delta \geq 0$,
17.	$\varsigma = 0 \Rightarrow u = 0 \wedge s_c = 0 \wedge b\mu = 0$.

Table 7.10: Invariants of system $S(\overline{0})$.

and 406 time-lock alternatives. Most of the conditions, guarding these actions are never satisfied when starting in the initial state. This overhead is reduced when we work using the invariants listed in Table 7.10.

The invariants of Table 7.10 furthermore show that parameter σ is redundant, as its value can be recovered via parameter ς. Similarly, parameters t, s_b, x_b and ϱ are redundant. Removing these redundant parameters, we arrive at a specification for the belt system $S(\overline{0})$, given by Table 7.11. The terms, resulting from encapsulating and expanding are listed in Appendix A. The absence of time-locks in Table 7.11 is explained by the fact that we are able to remove all time-locks appearing in Appendix A.

$$
\begin{array}{l}
\textbf{proc } S = \bar{c} \lessdot 0 \cdot S_0(0,0,0) \\[1em]
\textbf{proc } S_0(u{:}\mathbb{R}_{\geq \mathbf{0}}, s{:}\mathbb{R}, \mu \in [-T_{max}, T_{max}]) = \\
\quad \sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \mathbf{a} \lessdot (u+d) \cdot S^{-}_{1,-}(u+d, b\mu d + s, 0, \mu) \\
\qquad \lhd\, s \geq 0 \wedge 0 \leq s + b\mu d \leq v_{max} \rhd \delta \lessdot \mathbf{0} \\
+\mathbf{F}(0) \lessdot (u + \frac{-s}{b\mu}) \cdot S_0(u + \frac{-s}{b\mu}, 0, 0) \\
\qquad \lhd\, b\mu \neq 0 \rhd \delta \lessdot \mathbf{0} \\[1em]
\textbf{proc } S^{+}_0(u, \Delta{:}\mathbb{R}_{\geq \mathbf{0}}, s \in \mathbb{R}, \mu \in [-T_{max}, T_{max}]) = \\
\quad \sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \mathbf{a} \lessdot (u+d) \cdot S^{+}_{1,-}(u+d, b\mu d + s, 0, \mu) \\
\qquad \lhd\, 0 \leq s + b\mu d \leq v_{max} \wedge d \leq \Delta \rhd \delta \lessdot \mathbf{0} \\
+\mathbf{F}(0) \lessdot (u + \frac{v_{max}-s}{b\mu}) \cdot S^{+}_0(u + \frac{v_{max}-s}{b\mu}, \Delta - \frac{v_{max}-s}{b\mu}, v_{max}, 0) \\
\qquad \lhd\, b\mu \neq 0 \wedge \frac{v_{max}-s}{b\mu} \leq \Delta \rhd \delta \lessdot \mathbf{0} \\
+\mathbf{F}(-T_{max}) \lessdot (u + \Delta) \cdot S_0(u+\Delta, b\mu\Delta + s, -T_{max}) \\
\qquad \lhd\, 0 \leq b\mu\Delta + s \leq v_{max} \rhd \delta \lessdot \mathbf{0} \\[1em]
\textbf{proc } S^{-}_{1,-}(u{:}\mathbb{R}_{\geq \mathbf{0}}, s, x{:}\mathbb{R}, \mu \in [-T_{max}, T_{max}]) = \\
\quad \mathbf{F}(0) \lessdot u \cdot S^{+}_{1,-}(u, s, x, 0) \\
\qquad \lhd\, s = v_{max} \rhd \delta \lessdot \mathbf{0} \\
+\mathbf{F}(T_{max}) \lessdot u \cdot S^{+}_{1,-}(u, s, x, T_{max}) \\
\qquad \lhd\, s \neq v_{max} \rhd \delta \lessdot \mathbf{0} \\[1em]
\textbf{proc } S^{+}_{1,-}(u{:}\mathbb{R}_{\geq \mathbf{0}}, s, x{:}\mathbb{R}, \mu \in [-T_{max}, T_{max}]) = \\
\quad \mathbf{F}(0) \lessdot (u + \frac{v_{max}-s}{b\mu}) \cdot S^{+}_{1,-}(u + \frac{v_{max}-s}{b\mu}, v_{max}, \frac{1}{2}b\mu(\frac{v_{max}-s}{b\mu})^2 + s\frac{v_{max}-s}{b\mu} + x, 0) \\
\qquad \lhd\, 0 \leq \frac{1}{2}b\mu(\frac{v_{max}-s}{b\mu})^2 + s\frac{v_{max}-s}{b\mu} + x \leq \frac{1}{2}l \wedge b\mu \neq 0 \rhd \delta \lessdot \mathbf{0} \\
+\bar{c} \lessdot (u + \frac{l-2x}{2s}) \cdot S_{1,+}(u + \frac{l-2x}{2s}, v_{max}, \frac{1}{2}l, \mu) \\
\qquad \lhd\, b\mu = 0 \wedge s \geq 0 \rhd \delta \lessdot \mathbf{0} \\
+\bar{c} \lessdot (u + \frac{-s+\sqrt{s^2 - b\mu(2x-l)}}{b\mu}) \cdot S_{1,+}(u + \frac{-s+\sqrt{s^2-b\mu(2x-l)}}{b\mu}, \sqrt{s^2 - b\mu(2x-l)}, \frac{1}{2}l, \mu) \\
\qquad \lhd\, b\mu \neq 0 \wedge \sqrt{s^2 - b\mu(2x-l)} \leq v_{max} \rhd \delta \lessdot \mathbf{0} \\[1em]
\textbf{proc } S_{1,+}(u{:}\mathbb{R}_{\geq \mathbf{0}}, s, x{:}\mathbb{R}, \mu \in [-T_{max}, T_{max}]) = \\
\quad \sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \mathbf{a} \lessdot (u+d) \cdot S_2(u+d, s + b\mu d, 0, \frac{1}{2}b\mu d^2 + sd + x, \mu) \\
\qquad \lhd\, 0 \leq s + b\mu d \leq v_{max} \wedge 0 \leq \frac{1}{2}b\mu d^2 + sd + x \leq l \rhd \delta \lessdot \mathbf{0} \\
+\mathbf{d} \lessdot (u + \frac{-s+\sqrt{s^2-2b\mu(x-l)}}{b\mu}) \cdot S^{+}_0(u + \frac{-s+\sqrt{s^2-2b\mu(x-l)}}{b\mu}, \mathfrak{d}, \sqrt{s^2 - 2b\mu(x-l)}, \mu) \\
\qquad \lhd\, b\mu \neq 0 \wedge \sqrt{s^2 - 2b\mu(x-l)} \leq v_{max} \rhd \delta \lessdot \mathbf{0} \\
+\mathbf{d} \lessdot (u + \frac{l-x}{s}) \cdot S^{+}_0(u + \frac{l-x}{s}, \mathfrak{d}, s, \mu) \\
\qquad \lhd\, b\mu = 0 \wedge 0 \leq s \leq v_{max} \delta \lessdot \mathbf{0} \\
+\mathbf{F}(0) \lessdot (u + \frac{v_{max}-s}{b\mu}) \cdot S_{1,+}(u + \frac{v_{max}-s}{b\mu}, v_{max}, \frac{1}{2}b\mu(\frac{v_{max}-s}{b\mu})^2 + s\frac{v_{max}-s}{b\mu} + x, 0) \\
\qquad \lhd\, \frac{1}{2}b\mu(\frac{v_{max}-s}{b\mu})^2 + s\frac{v_{max}-s}{b\mu} + x \leq l \wedge b\mu \neq 0 \rhd \delta \lessdot \mathbf{0}
\end{array}
$$

Table 7.11: Belt System after Expansion

$$
\begin{array}{l}
\textbf{proc}\ S_2(u{:}\mathbb{R}_{\geq \mathbf{0}}, s, x, y{:}\mathbb{R}, \mu \in [-T_{\mathit{max}}, T_{\mathit{max}}]) = \\
\quad \mathbf{d}\,\text{ʿ}(u + \frac{-s+\sqrt{s^2-2b\mu(y-l)}}{b\mu}) \cdot S^{-}_{1,+}(u + \frac{-s+\sqrt{s^2-2b\mu(y-l)}}{b\mu}, \sqrt{s^2 - 2b\mu(y-l)}, x + l - y, \mu) \\
\qquad \lhd\, b\mu \neq 0 \wedge \sqrt{s^2 - 2b\mu(y-l)} \leq v_{\mathit{max}} \rhd \delta\,\text{ʿ}\mathbf{0} \\
+\mathbf{d}\,\text{ʿ}(u + \frac{l-y}{s}) \cdot S^{-}_{1,+}(u + \frac{l-y}{s}, s, x + l - y, \mu) \\
\qquad \lhd\, b\mu = 0 \wedge 0 \leq s \leq v_{\mathit{max}} \rhd \delta\,\text{ʿ}\mathbf{0} \\
+\mathbf{F}(0)\,\text{ʿ}(u + \frac{v_{\mathit{max}}-s}{b\mu})\cdot \\
\quad S_2(u + \frac{v_{\mathit{max}}-s}{b\mu}, v_{\mathit{max}}, \frac{1}{2}b\mu(\frac{v_{\mathit{max}}-s}{b\mu})^2 + s\frac{v_{\mathit{max}}-s}{b\mu} + x, \frac{1}{2}b\mu(\frac{v_{\mathit{max}}-s}{b\mu})^2 + s\frac{v_{\mathit{max}}-s}{b\mu} + y, 0) \\
\qquad \lhd\, b\mu \neq 0 \wedge 0 \leq \frac{1}{2}b\mu(\frac{v_{\mathit{max}}-s}{b\mu})^2 + s\frac{v_{\mathit{max}}-s}{b\mu} + x \leq \\
\qquad\ \ \frac{1}{2}b\mu(\frac{v_{\mathit{max}}-s}{b\mu})^2 + s\frac{v_{\mathit{max}}-s}{b\mu} + y \leq l \rhd \delta\,\text{ʿ}\mathbf{0}
\end{array}
$$

Table 7.11: Belt System after Expansion

7.3.4 System Properties

The processes listed in Table 7.11 characterise the behaviour of the conveyor belt under direct control of our controller. These processes are distilled from the terms found in Appendix A. The annotations of the numerous time-locks, also present in Appendix A show they can all be removed, meaning that there is no conflicting situation in the parallel system. The belt system itself is easily seen to be free of deadlock. The invariants we used for reducing the complexity of encapsulating and expanding have the additional purpose of characterising the reachable state space of the belt system. This means that all reachable states satisfy the invariant $0 \leq s_b \leq s_c \leq v_{\mathit{max}}$, meaning that the velocity of the belt is always within the allowed range of 0 and v_{max}. The fact that no deadlock occurs illustrates there are never more than two trays on the belt. Moreover, whenever there are two trays on the belt ($\varsigma = 6$), invariant $y_b - x_b \geq \frac{1}{2}l$ proves the distance between both trays is at least $\frac{1}{2}l$. Finally, from the invariant $\varsigma \notin \{1,3\} \wedge b\mu = 0 \Rightarrow s_c = v_{\mathit{max}}$, we can draw the conclusion that the belt is accelerated once a tray has been signalled, meaning that the tray is eventually delivered.

7.4 Mathematical Analysis of the Belt System

The conveyor belt system lends itself perfectly for a performance analysis. Such an analysis, however, is beyond the reach of our capabilities, as it requires the introduction of stochastic information into the models of the conveyor belt and its environment. The probabilistic extensions of ACP (see e.g. [10]), may provide an interesting starting point for extending μCRL$_t$ to a stochastic process algebra. In the remainder of this section, we restrict ourselves to a mathematical analysis of the belt. We study the belt system in several different environments. Issues we address are the throughput time per tray and the time-velocity characteristics.

The environments we consider are slightly simplified reflections of reality. This allows us to study the belt system in well-understood situations.

7.4.1 Continuous Delivery of Trays

To study the extremes of our system, we first investigate the environment in which the conveyor belt is the system that forms the bottleneck. This means that the up-stream environment is constantly able to add trays to the belt and does so instantaneously. By synchronising on the $\bar{c}$ action, issued by the controller, the up-stream environment is notified it can issue another tray. The process, modelling this environment is specified in Eqn. (7.6).

$$\textbf{proc}\ E(t{:}\mathbb{R}_{\geq \mathbf{0}}) = \sum_{d:\mathbb{R}_{\geq \mathbf{0}}} c^{\text{c}}(t+d) \cdot a_e{}^{\text{c}}(t+d) \cdot E(t+d) \tag{7.6}$$

The action a_e denotes the event of delivering a tray on the belt, whereas the action c is used for receiving the notification on acceptance of new trays on the belt. The communications are defined in Table 7.12. The resulting system S_e is specified by Eqn.(7.7).

Belt System	Environment	Communication
$\mathbf{a}$	a_e	$\mathbf{a}_e$
$\bar{c}$	c	$\mathbf{c}$

Table 7.12: Communications

$$S_e = \partial_{\{\mathbf{a},a_e,c,\bar{c}\}}(S \| E(\mathbf{0})) \tag{7.7}$$

It is straightforward to see system S_e stabilises after some initial time t' (the start-up time). At start-up, the belt is at a stand-still; since trays arrive immediately when allowed, there is no time for the belt to decelerate, and hence, the time it takes for the belt to reach maximal velocity is $T_s = \frac{v_{max}}{bT_{max}}$. The average throughput of the belt system is the average time it takes for the belt to transport a tray from front to back; in this case this is easily seen to be $T_t = \frac{l}{v_{max}}$.

This throughput time and the start-up time are also optimal in the sense that there is no other utilisation of the belt system that outperforms system S_e, simply because there is no other environment that can ship trays on the belt with such a high frequency. Therefore, we can use the times T_s and T_t for purposes of comparison with other systems.

Since system S_e does not depend on the constant $\eth$ (i.e. apart from at start-up, the belt is always occupied by at least one tray) the energy efficiency is 100%.

Although this system is optimal with respect to the start-up noise time and the throughput time, it is also in general not very realistic, since in practice there is hardly ever an infinite buffer containing the trays.

7.4.2 Periodic Delivery of Trays

A more common environment is the environment in which trays are shipped to the belt periodically. This period T, represents the time between two successive arrivals of trays on the belt. The up-stream environment can in this case be modelled as a buffer of infinite capacity that outputs trays every other T^{th} time-unit, see Eqn. (7.8).

$$\mathbf{proc}\; E(t{:}\mathbb{R}_{\geq \mathbf{0}}) = \sum_{d:\mathbb{R}_{\geq \mathbf{0}}} c^{\varsigma}(t+d) \cdot a_e{}^{\varsigma}(t+T) \cdot E(t+T) \tag{7.8}$$

The communications between the belt system and the environment are again as defined by Table 7.12. The resulting system $S_{e'}$ is specified by Eqn. (7.9).

$$S_{e'} = \partial_{\{\mathbf{a}, a_e, c, \bar{c}\}}(S \| E(\mathbf{0})) \tag{7.9}$$

Some insight in the belt system immediately reveals the above system can potentially deadlock. This happens when the period T is chosen too small, in which case the up-stream environment receives a notification of clearance of the belt from the controller long after the period T has passed. This situation is easily avoided by choosing a safe lower bound for the period T.

Establishing a Lower Bound. A lower bound for T can be established by determining the maximal time it takes to transport a tray half-way the belt. Observe that the maximal time is only needed when starting from a stand-still. We can distinguish two cases: either the belt is accelerated to v_{max} before or after the tray is transported half-way the belt. In case the belt does not reach v_{max}, this lower bound is given by $\sqrt{\frac{l}{bT_{max}}}$. In case the belt reaches v_{max} before the tray is transported half-way the belt, the lower bound is determined by $\frac{1}{2}(\frac{l}{v_{max}} + \frac{v_{max}}{bT_{max}})$. Note that, since all constants are strictly positive, these bounds are well-defined. Moreover, following general rules of calculus, we know $\sqrt{\frac{l}{bT_{max}}} \leq \frac{1}{2}(\frac{l}{v_{max}} + \frac{v_{max}}{bT_{max}})$. Hence, our lower bound for T is $\frac{1}{2}(\frac{l}{v_{max}} + \frac{v_{max}}{bT_{max}})$.

We split our analysis in two cases: the belt eventually reaches the maximal velocity v_{max}, and the belt never reaches maximal velocity. Both cases are dealt with separately.

Maximal velocity is never reached

This situation is likely to happen when the period T with which the trays are put on the belt is large enough for the belt to always be decelerated below some critical velocity v_c before a new tray is put on the belt. This situation is therefore possible only when there is at most one tray on the belt. Fig. 7.5 shows the evolution of the velocity. The starting velocity for the i^{th} tray is given by s_i. The maximal velocity the belt reaches after delivery of the i^{th} tray is given by r_i and the time it takes to deliver the i^{th} tray to the down-stream environment is t_i. Based on our knowledge of the continuous model, we can derive the following difference equations:

$$\begin{aligned} r_i &= s_i + bT_{max}(t_i + \mathfrak{d}) \\ s_{i+1} &= r_i - bT_{max}(T - t_i - \mathfrak{d}) \\ t_i &= \frac{-s_i + \sqrt{s_i^2 + 2bT_{max}l}}{bT_{max}} \end{aligned} \tag{7.10}$$

The above difference equations allow us to derive a recurrence relation for the floor velocity s_{i+1}. For convenience sake, we abbreviate bT_{max} with a, and require $a > 0$.

$$s_{i+1} = 2\sqrt{s_i^2 + 2al} - s_i + 2a\mathfrak{d} - aT \tag{7.11}$$

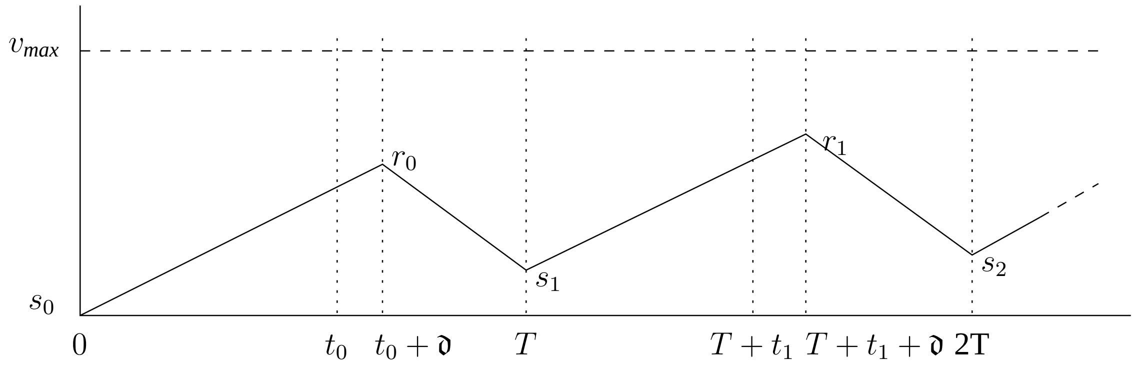


Figure 7.5: Time-Velocity Diagram

Boundary conditions. We analyse the belt system under the assumption that for all s_i, r_i, we have $0 \leq s_i \leq r_i \leq v_{max}$. This means, restrictions to the constants $a, \mathfrak{d}, T, l$ and v_{max} apply. We proceed by investigating these restrictions. We first reduce the assumption $0 \leq s_i \leq r_i \leq v_{max}$ to an assumption on s_i only. Since $r_i = s_i + at_i + a\mathfrak{d}$, we can replace $r_i \leq v_{max}$ with $s_i \leq \sqrt{(v_{max} - a\mathfrak{d})^2 - 2al}$, provided that $v_{max} \geq a\mathfrak{d} + \sqrt{2al}$. We refer to the upper bound $\sqrt{(v_{max} - a\mathfrak{d})^2 - 2al}$ as the *critical upper bound* and abbreviate it with v_c. Obviously, $v_c \geq 0$.

We assume that $0 \leq s_i \leq v_c$, and analyse the requirements to maintain $0 \leq s_{i+1} \leq v_c$. Under the proviso $T \geq 2\mathfrak{d}$, this is rewritten to $s_i + aT - 2a\mathfrak{d} \leq 2\sqrt{s_i^2 + 2al} \leq s_i + v_c + aT - 2a\mathfrak{d}$, which follows from the conjunction of $3s_i^2 + 2(aT - 2a\mathfrak{d})s_i - (aT - 2a\mathfrak{d})^2 + 8al \geq 0$ and $0 \geq 3s_i^2 + 2(v_c - 2a\mathfrak{d} + aT)s_i - (v_c - 2a\mathfrak{d} + aT)^2 + 8al$.

We analyse both situations separately. From $3s_i^2 + 2(aT - 2a\mathfrak{d})s_i - (aT - 2a\mathfrak{d})^2 + 8al \geq 0$, we can deduce that this holds unconditionally for $l \geq \frac{1}{6}a(T - 2\mathfrak{d})^2$. A tighter bound is obtained by solving for which s_i the above inequality is satisfied. It turns out that it is satisfied for both $s_i \leq \frac{1}{3}(aT - 2a\mathfrak{d} - 2\sqrt{(aT - 2a\mathfrak{d})^2 - 6al})$, or $s_i \geq \frac{1}{3}(aT - 2a\mathfrak{d} + 2\sqrt{(aT - 2a\mathfrak{d})^2 - 6al})$. The second solution cannot be brought into accord with our assumption $0 \leq s_i \leq v_c$. For the first solution, we calculate $\frac{1}{3}(aT - 2a\mathfrak{d} - 2\sqrt{(aT - 2a\mathfrak{d})^2 - 6al}) \geq 0$, yielding the tighter bound $l \geq \frac{1}{8}a(T - 2\mathfrak{d})^2$.

From $0 \geq 3s_i^2 + 2(v_c - 2a\mathfrak{d} + aT)s_i - (v_c - 2a\mathfrak{d} + aT)^2 + 8al$, we can deduce that for all s_i, for which $\frac{1}{3}(v_c - 2a\mathfrak{d} + aT - 2\sqrt{(v_c - 2a\mathfrak{d} + aT)^2 - 6al}) \leq s_i$ and $s_i \leq \frac{1}{3}(v_c - 2a\mathfrak{d} + aT + 2\sqrt{(v_c - 2a\mathfrak{d} + aT)^2 - 6al})$. Bringing this requirement into accord with the assumption $0 \leq s_i \leq v_c$, yields the following inequalities. From $v_c - 2a\mathfrak{d} + aT - 2\sqrt{(v_c - 2a\mathfrak{d} + aT)^2 - 6al} \leq 0$, we derive $v_c \leq -(aT - 2a\mathfrak{d}) - 2\sqrt{2al}$ and $v_c \geq 2\sqrt{2al} - (aT - 2a\mathfrak{d})$. The first solution is invalidated by our prior assumptions. Second, we analyse $v_c \leq \frac{1}{3}(v_c - 2a\mathfrak{d} + aT + 2\sqrt{(v_c - 2a\mathfrak{d} + aT)^2 - 6al})$, leading to the requirement $v_c \geq \frac{2l}{T-2\mathfrak{d}} - \frac{1}{4}a(T - 2\mathfrak{d})$.

An inventory of the above requirements learns we can manage to invariantly have $0 \leq s_i \leq r_i \leq v_{max}$ if the requirements, listed in Table 7.13 are satisfied.

Equilibrium. Now, provided that the constants $a, \mathfrak{d}, l, T$ and v_{max} satisfy the above-derived conditions, we further investigate the behaviour of the belt system. Obviously, we have $0 \leq s_i \leq r_i \leq v_{max}$. Questions that arise concern the stability of the belt system. We address this question by investigating the equilibrium of the system. Finding an equilibrium is straightforward, as it requires the computation of a velocity v_*, such that $s_i = s_{i+1} = v_*$. Using Eqn. (7.11), we arrive

$$\begin{array}{l}
T \geq 2\mathfrak{d} \\
l \geq \frac{1}{8}a(T-2\mathfrak{d})^2 \\
a\mathfrak{d} + \sqrt{(aT - 2a\mathfrak{d} - 2\sqrt{2al})^2 + 2al} \leq v_{max} \\
a\mathfrak{d} + \frac{2l}{T-2\mathfrak{d}} + \frac{1}{4}a(T-2\mathfrak{d}) \leq v_{max}
\end{array}$$

Table 7.13: Requirements

at the following solution:

$$v_* = \frac{2l}{T-2\mathfrak{d}} - \frac{1}{4}a(T-2\mathfrak{d}), \text{ provided that } T > 2\mathfrak{d} \tag{7.12}$$

Local Stability. It is well possible this equilibrium may never be reached. In fact, the belt system behaviour could be quite erratic, even close to the equilibrium. We prove, however, that this is not the case, as the behaviour close to the equilibrium is locally stable. This means that the closer we get to the equilibrium, the more the successive velocities tend towards v_*, although it may never really reach it.

We proceed as follows. We define a function $f(v) = 2\sqrt{v^2+2al} - v + 2a\mathfrak{d} - aT$. By definition, our equilibrium is locally stable if $|\frac{df}{dv}(v_*)| < 1$. A straightforward calculation yields $f'(v) = \frac{v}{\sqrt{v^2+2al}} - 1$. Since all velocities we consider are positive, we know for all $v > 0$, that $|f'(v)| < 1$. This immediately tells us that the equilibrium is locally stable.

Periodic Equilibrium. The equilibrium, calculated in the previous paragraphs, is stable only in its immediate neighbourhood. Therefore, other equilibriums could be found that, unlike the equilibrium discussed above, are only reached periodically. For these equilibriums, we can calculate a velocity $v_\#$, such that, after n transportations of trays, the belt again runs at velocity $v_\#$. For a two-cycle, we compute $s_{i+2} = s_i$, yielding the following solution:

$$v_\# = \frac{1}{2}\sqrt{(aT - 2a\mathfrak{d}) - 8al}, \text{ provided that } T \geq \frac{2}{a}\sqrt{2al} + 2\mathfrak{d} \tag{7.13}$$

The proviso, however, can be traced back to $l \leq \frac{1}{8}(T-2a\mathfrak{d})^2$. Since we already have requirement $l \geq \frac{1}{8}(T-2a\mathfrak{d})^2$, this means two-cycles only occur when l is exactly $\frac{1}{8}(T-2a\mathfrak{d})^2$. For this length, however, we can compute that $s_i = 0$ for all $i \geq 0$, and hence, $v_\#$ coincides with v_*.

Example 7.4.1. We analyse a concrete system. The system parameters are given in centimetres and seconds, where $a = 5$, $T = 20$, and $l = 160$. The plots of Figs. 7.6 to 7.9 represent the values of s_i and r_i, with increasing values of $\mathfrak{d}$. The data supports the findings in Table 7.13, as well as the predicted equilibrium of (7.12). Fig. 7.6 clearly shows that for $\mathfrak{d} = 1$ second, the behaviour of the belt is not in the range of the systems we study here. Solving the inequalities $\mathfrak{d} \leq \frac{1}{2}T$ and $l \geq \frac{1}{8}a(T-2\mathfrak{d})^2$ for $\mathfrak{d}$ learns that the systems we study satisfy $2 \leq \mathfrak{d} \leq 10$. The reader is invited to calculate lower bounds for the maximal velocity v_{max} for the here considered parameters.

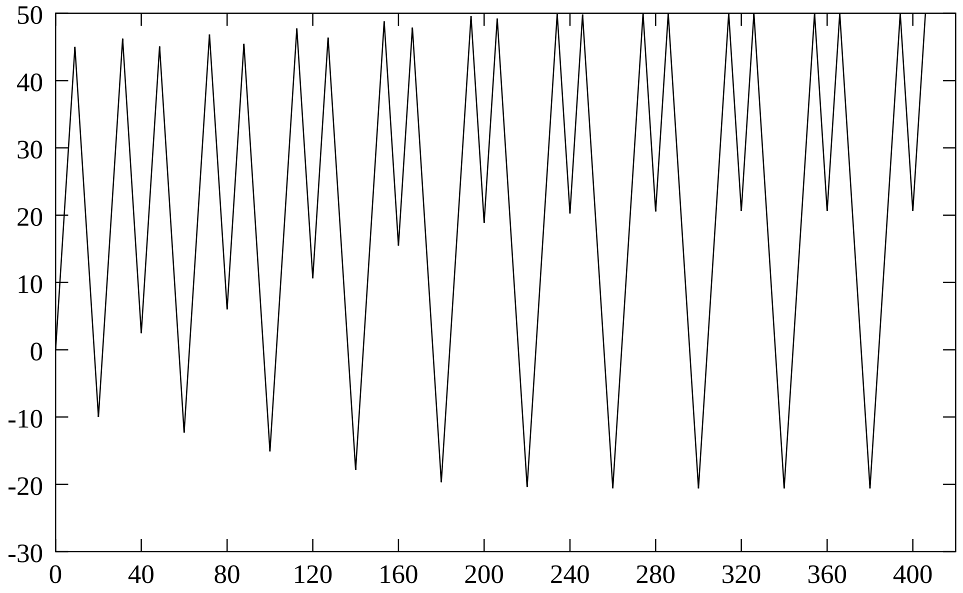


Figure 7.6: Velocity-Time diagram for $\mathfrak{d} = 1$

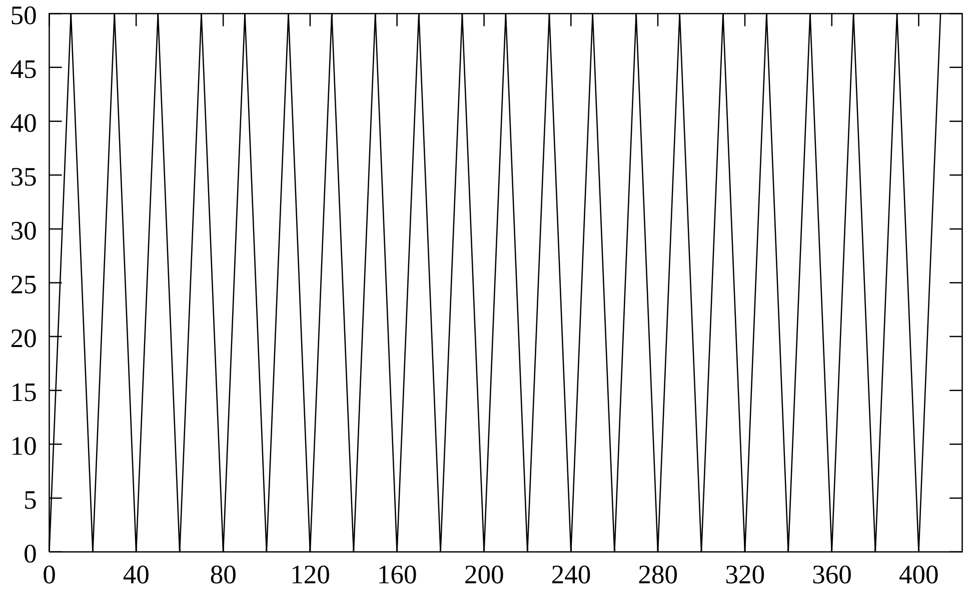


Figure 7.7: Velocity-Time diagram for $\mathfrak{d} = 2$

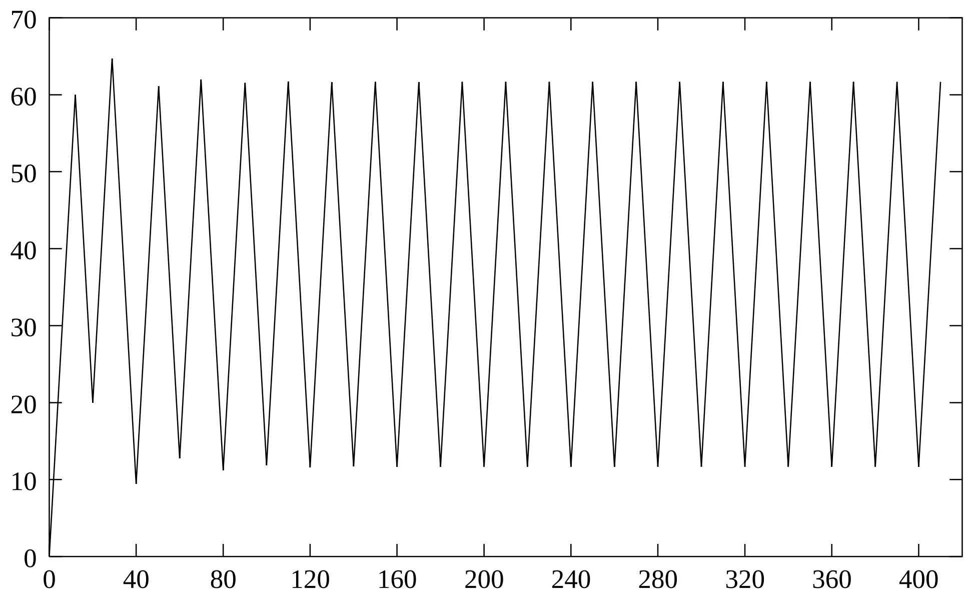


Figure 7.8: Velocity-Time diagram for $\mathfrak{d} = 4$

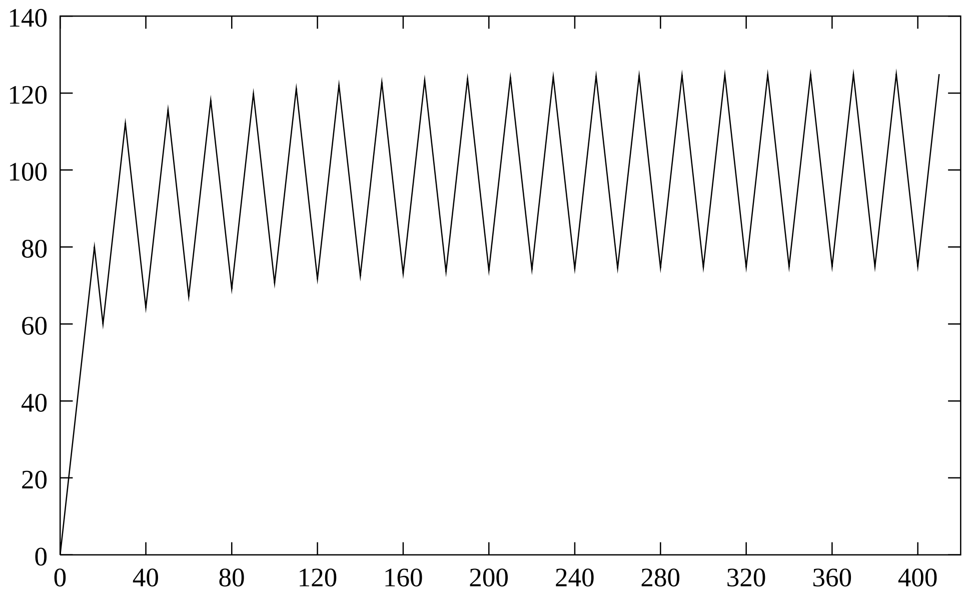


Figure 7.9: Velocity-Time diagram for $\mathfrak{d} = 8$

Throughput Time. We calculate the throughput time for the systems after stabilising to the equilibrium velocity. Clearly, the time it takes to deliver a tray to the belt's down-stream environment is $t_* = \frac{1}{2}T - \mathfrak{d}$. Comparing this throughput time with the optimal throughput time $T_t = \frac{l}{v_{max}}$, yields the ratio $\frac{2}{l}(T - 2\mathfrak{d})v_{max}$.

Example 7.4.2. Figure 7.10 depicts the situation where $a = 5$, $l = 160$, $\mathfrak{d}$ and T are variable and the speed v_{max} is such that it is the maximal speed that can be reached, based on the parameters $a, l, \mathfrak{d}$ and T. The z-axis of Fig. 7.10 shows the throughput efficiency of the belt system, relative to the maximal throughput efficiency, i.e. $100\frac{T_t}{t_*}$.

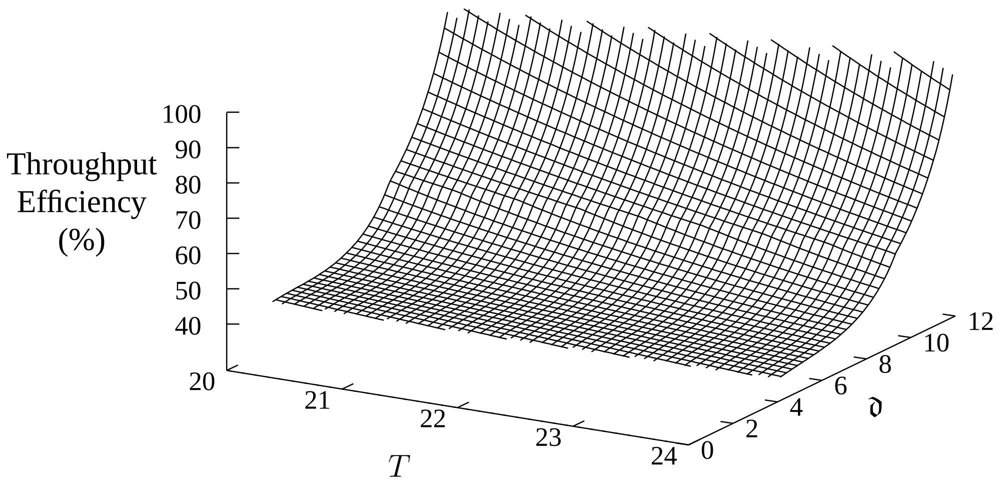


Figure 7.10: Throughput Efficiency

Maximal velocity is reached

In many cases, the maximal velocity is reached during the transportation of a tray. If, during this period, another tray is put on the belt, the belt will continue running at maximal speed. Here, however, we restrict ourselves to the situation where the belt is decelerated (or about to decelerate) before another tray arrives, and is accelerated before, or exactly on the moment the belt arrives at a stand-still. The systems we analyse here always reach maximal velocity during the transportation of a single tray. This means that there are several requirements on the maximal velocity v_{max}. For instance, the belt must reach v_{max} starting from a stand-still, such that the time needed to do so, minus the delay time $\mathfrak{d}$, is at most the time needed to cover a distance of l length-units. This yields the mathematical inequality $l \geq \frac{v_{max} - a\mathfrak{d}}{2a}$, or rephrased to a requirement on v_{max}, we assume $v_{max} \in [a\mathfrak{d} - \sqrt{2al}, a\mathfrak{d} + \sqrt{2al}]$. Note that there may be a class of belt systems for which only occasionally the maximal velocity is reached; such systems fall outside the scope of the analysis described here.

The velocity-time diagrams of the belt systems we study is typically described by Fig. 7.11. Such systems are likely to have low maximal velocities v_{max} in combination with relatively large delays $\mathfrak{d}$. Figure 7.11 refers to several points in time; it takes t_i time-units to accelerate the belt to v_{max}, starting from velocity s_i. The tray is then delivered after t'_i time-units (measured from the moment maximal velocity is reached), etc. Based on our knowledge of the continuous model

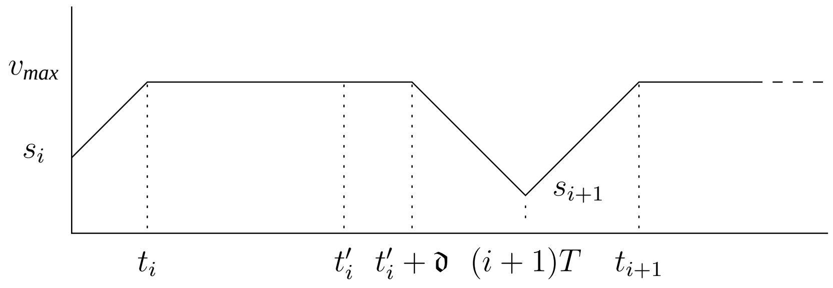


Figure 7.11: *Time-Velocity diagram*

and the controller of the belt system, the above situation is described by the following difference equations:

$$\begin{aligned} t_i &= \tfrac{v_{max} - s_i}{a} \\ t'_i &= \tfrac{2l - at_i^2 - 2s_i t_i}{2v_{max}} \\ s_{i+1} &= v_{max} - a(T - t_i - t'_i - \mathfrak{d}) \end{aligned} \tag{7.14}$$

Notice that t_i and t'_i are elements of the time-domain, and are therefore non-negative. This yields two requirements, viz. $t_i \geq 0$ and $t'_i \geq 0$, which we analyse below. Based on the equations in (7.14) we derive a recurrence relation for s_{i+1}, see Eqn. (7.15).

$$s_{i+1} = \frac{s_i^2 - 2v_{max}s_i + 3v_{max}^2 - 2av_{max}(T - \mathfrak{d}) + 2al}{2v_{max}} \tag{7.15}$$

Boundary conditions. We conduct the analysis under the assumption that for all s_i we have $0 \leq s_i \leq v_{max}$. We first investigate the requirements we have on v_{max} that guarantees we have $t_i \geq 0$ and $t'_i \geq 0$. The first is readily satisfied under the assumption that $0 \leq s_i \leq v_{max}$. For the second inequality, we derive $0 \geq 2s_i^2 - 4v_{max}s_i + v_{max}^2 - 2al$. This is satisfied whenever $s_i \in [v_{max} - \sqrt{\frac{1}{2}v_{max}^2 + al}, v_{max} + \sqrt{\frac{1}{2}v_{max}^2 + al}]$. Since we know $0 \leq s_i \leq v_{max}$, the derived inequality must contain the interval $[0, v_{max}]$. This obviously holds for v_{max}. For the left bound 0, we derive the inequality $v_{max} \leq \sqrt{2al}$.

We continue to investigate the requirements leading to $0 \leq s_{i+1} \leq v_{max}$, under the assumption that $0 \leq s_i \leq v_{max}$. Rewriting $0 \leq s_{i+1} \leq v_{max}$ leads to the inequality $0 \leq s_i^2 - 2v_{max}s_i + 3v_{max}^2 - 2av_{max}(T - \mathfrak{d}) + 2al \leq 2v_{max}^2$. We solve these two inequalities separately.

For $0 \leq s_i^2 - 2v_{max}s_i + 3v_{max}^2 - 2av_{max}(T - \mathfrak{d}) + 2al$, we derive two inequalities; either $s_i \leq v_{max} - \sqrt{2a(T - \mathfrak{d})v_{max} - 2v_{max}^2 - 2al}$ or $s_i \geq v_{max} + \sqrt{2a(T - \mathfrak{d})v_{max} - 2v_{max}^2 - 2al}$. Obviously, this latter solution is in violation with our assumption that $s_i \leq v_{max}$. We bring the first solution into accord with the assumption that $0 \leq s_i$. Thereto, we solve the inequality $0 \leq v_{max} - \sqrt{2a(T - \mathfrak{d})v_{max} - 2v_{max}^2 - 2al}$. This leads to the requirement that we have either $v_{max} \leq \frac{1}{3}(aT - a\mathfrak{d} - \sqrt{(aT - a\mathfrak{d})^2 - 6al})$ or $v_{max} \geq \frac{1}{3}(aT - a\mathfrak{d} + \sqrt{(aT - a\mathfrak{d})^2 - 6al})$.

We proceed by investigating $s_i^2 - 2v_{max}s_i + 3v_{max}^2 - 2av_{max}(T - \mathfrak{d}) + 2al \leq 2v_{max}^2$. For this, we derive s_i lies in the interval $[v_{max} - \sqrt{2a(T - \mathfrak{d})v_{max} - 2al}, v_{max} + \sqrt{2a(T - \mathfrak{d})v_{max} - 2al}]$. This interval must be a superset of the interval $[0, v_{max}]$; it is obvious that it contains v_{max}. Therefore, we focus on $v_{max} - \sqrt{2a(T - \mathfrak{d})v_{max} - 2al} \leq 0$. This yields the additional requirement on v_{max}, i.e. $v_{max} \in [a(T - \mathfrak{d}) - \sqrt{(aT - a\mathfrak{d})^2 - 2al}, a(T - \mathfrak{d}) + \sqrt{(aT - a\mathfrak{d})^2 - 2al}]$. Summarising, the above requirements are listed in Table 7.14.

$v_{\mathit{max}} \leq \sqrt{2al}$
$v_{\mathit{max}} \in [a\mathfrak{d} - \sqrt{2al}, a\mathfrak{d} + \sqrt{2al}]$
$v_{\mathit{max}} \notin [\frac{1}{3}(a(T-\mathfrak{d}) - \sqrt{(aT-a\mathfrak{d})^2 - 6al}), \frac{1}{3}(a(T-\mathfrak{d}) + \sqrt{(aT-a\mathfrak{d})^2 - 6al})]$
$v_{\mathit{max}} \in [a(T-\mathfrak{d}) - \sqrt{(aT-a\mathfrak{d})^2 - 2al}, a(T-\mathfrak{d}) + \sqrt{(aT-a\mathfrak{d})^2 - 2al}]$

Table 7.14: Boundary Conditions

Equilibrium. Under the above circumstances, the belt system behaves more or less predictably. The equilibrium of the system is standardly computed as the fixed point of the recurrence relation, i.e. $v_* = s_{i+1} = s_i$ for some i. This leads to the solution, shown in Eqn. (7.16).

$$v_* = 2v_{\mathit{max}} - \sqrt{v_{\mathit{max}}^2 + 2av_{\mathit{max}}(T-\mathfrak{d}) - 2al} \tag{7.16}$$

Local Stability. The equilibrium, computed in the previous paragraph is locally stable. This is easily seen to be the case. We associate a function f to the recurrence relation for s_{i+1}, where f is defined as $f(v) = \frac{v^2 - 2v_{\mathit{max}}v + 3v_{\mathit{max}}^2 - 2av_{\mathit{max}}(T-\mathfrak{d}) + 2al}{2v_{\mathit{max}}}$. Then $\frac{df}{dv}(v) = \frac{2v - 2v_{\mathit{max}}}{2v_{\mathit{max}}}$. Then, it is easy to see that, provided that $T > \mathfrak{d} + \sqrt{\frac{l}{4a}}$, we have $|\frac{df}{dv}(v_*)| < 1$.

Periodic Equilibrium. We further investigate the existence of a periodic equilibrium. Here, we restrict our attention to finding an equilibrium for a two-cycle. This requires the solution for $s_{i+2} = s_i = v_{\#}$. One of the solutions matches the solution to the equilibrium velocity v_*, two of the solutions are outside the allowed range. The only viable solution is shown in Eqn. (7.17). This periodic equilibrium can only occur whenever $T \geq \frac{l}{v_{\mathit{max}}} + \frac{3v_{\mathit{max}}}{2a} + \mathfrak{d}$.

$$v_{\#} = \sqrt{2av_{\mathit{max}}(T-\mathfrak{d}) - 3v_{\mathit{max}}^2 - 2al} \tag{7.17}$$

Given this proviso, we can infer that the period T is large enough to transport a tray a distance of l length-units (which takes $\frac{l}{v_{\mathit{max}}}$ time-units), wait for $\mathfrak{d}$ time-units, and decelerate for $\frac{3v_{\mathit{max}}}{2a}$ time-units, taking the velocity below 0 in case the controller does not interfere. Hence, the periodic equilibrium does not apply to the systems we analyse here.

Example 7.4.3. As an example, we consider the belt system with parameters $a = 5$, $T = 20$, $l = 160$, where length-units are centimetres and time-units are seconds. Figures 7.12 to 7.14, show the behaviour of the belt system with resp. $\mathfrak{d} = 1$, $\mathfrak{d} = 2$ and $\mathfrak{d} = 3$. Interesting to see is that in Fig. 7.14, maximal velocity is maintained during a period of 37 seconds (i.e. from time 2 through 39). This is due to the delay of 3 seconds, causing the belt not to decelerate before the second tray arrives. Note that all systems eventually stabilise at the predicted velocity.

7.5 Summary

In this chapter, we used μCRL$_t$ to describe and analyse a hybrid system. The system consists of a multi-purpose conveyor belt, for which we specified a dedicated controller that ensures the

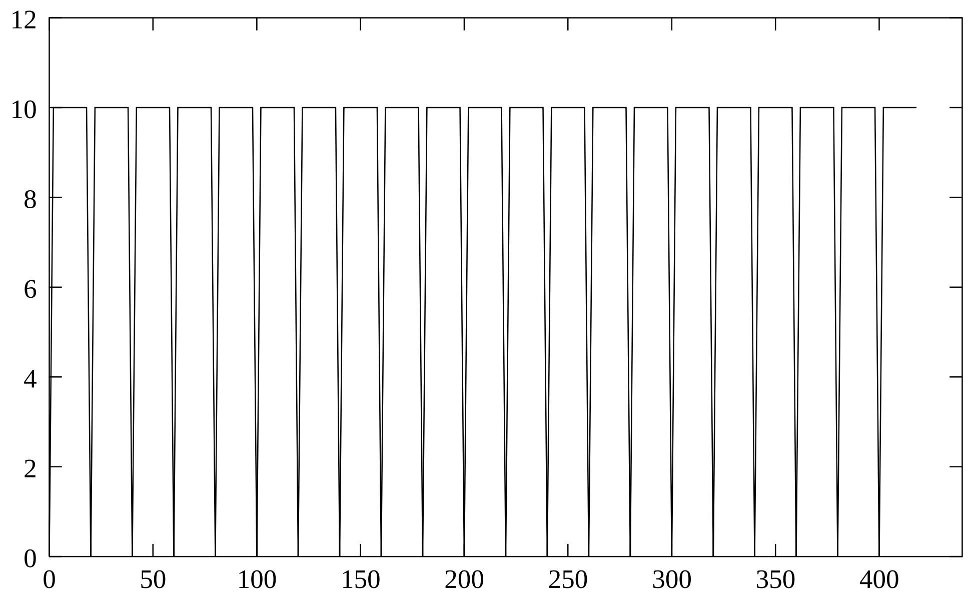


Figure 7.12: Velocity-Time diagram for $\mathfrak{d} = 1$

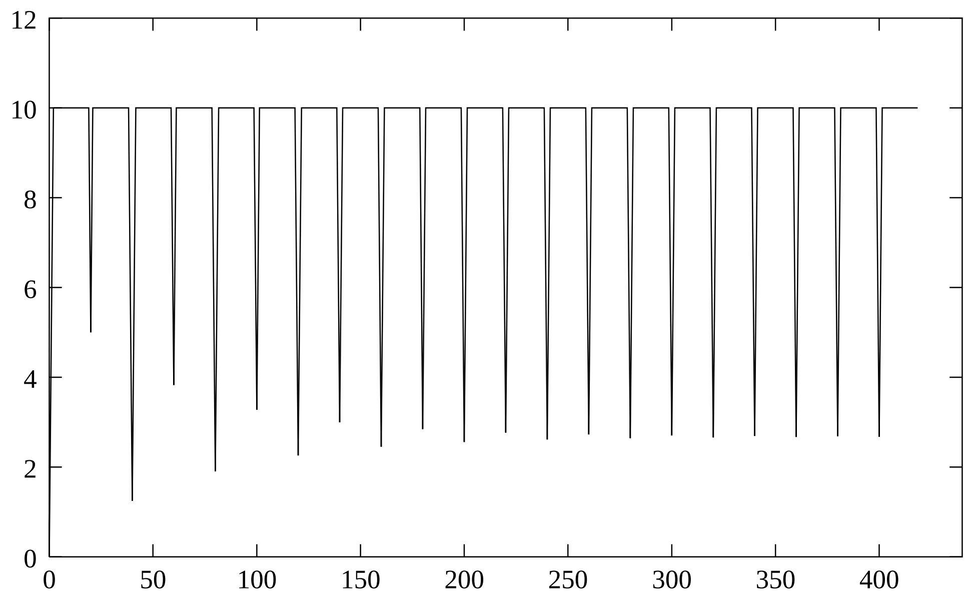


Figure 7.13: Velocity-Time diagram for $\mathfrak{d} = 2$

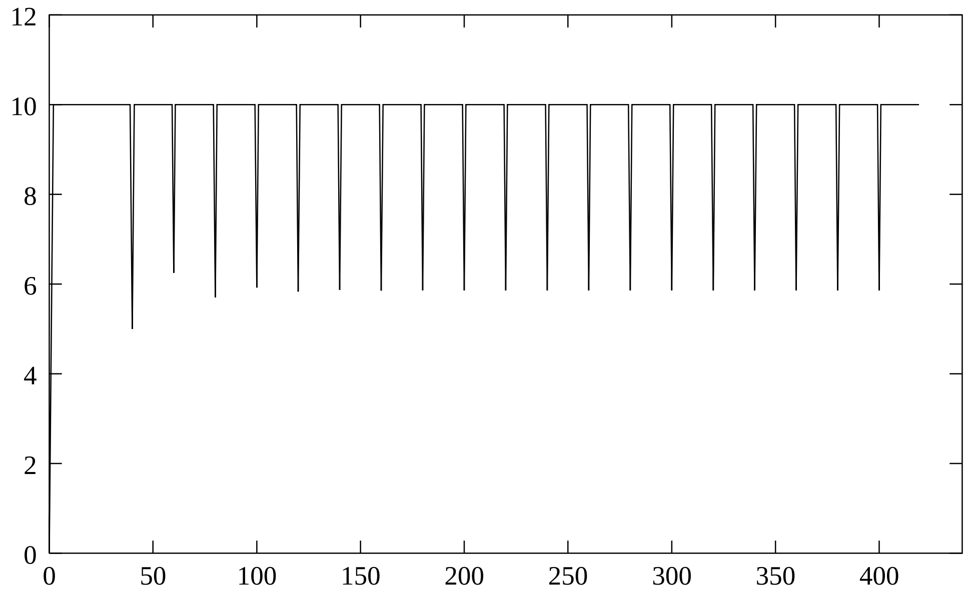


Figure 7.14: Velocity-Time diagram for $\mathfrak{d} = 3$

correct behaviour of the conveyor belt. Describing the components in μCRL$_t$ turns out to be fairly straightforward. Using techniques developed for μCRL$_t$, we have been able to verify the controller we specified does indeed ensure the conveyor belt behaves in line with the control objectives specified in the first section. The techniques, used in the verification are adaptations of techniques commonly used in verifications in the untimed setting. This suggests the design of μCRL$_t$ indeed supports, at least to some extent, the extension of proof techniques for μCRL to the timed setting. However, all that glitters is not gold. We observe that the verification, described in this chapter, requires more address than generally falls to the share of those who are unaccustomed to applying algebraic techniques. Section 7.3 and Appendix A illustrate the need for an almost diabolic precision and bookkeeping habit to successfully apply some of the techniques of μCRL$_t$ in practice. Note that these traits are typically the hall-marks of computers. An investment in the construction of tool support to alleviate this burden, therefore, seems logical. Of course, the desire to have tool support, assisting in, or even automating, (parts of) the verification is not founded on the sole observation we just made on bookkeeping, but is often judged self-evident.

An interesting follow up on this case study is to investigate the behaviour of a chain of conveyor belts, each with their own controller. Avoiding collisions of goods between belts largely depends on the exchange of information between controllers. These interactions between downstream and up-stream controllers required are expected to substantially increase the overall complexity. Given the complexity of the system we studied in this chapter, verifying such chains of conveyor belt with the current state of techniques seems impossible. The follow up study therefore could serve as a nice benchmark for testing (future) tool support.

Apart from the process algebraic verifications conducted in the first half of this chapter, the second half of this chapter briefly addresses questions of a more analytical nature. Thus, we note there is an apparent gap between these two halves of this chapter. In the description and process algebraic analysis, we clearly benefitted from the combination of the continuous and discrete models. For the analytical analysis, such claims cannot be made. This analysis is inspired almost entirely by the continuous model of the belt system. In some respect, this is odd, as the information, captured by the process algebraic models largely overlaps the information captured by the difference equations models we analysed in the second half of this chapter. An investigation into these similarities might reveal a systematic approach to conducting an analytical analysis of (a class of) process algebraic models. Obviously more research is needed to further these ideas.

Chapter 8

Conclusions

In Section 8.1, we give a brief overview of the results obtained in this thesis. For a more detailed discussion on the obtained results, we refer to the concluding sections of each chapter. For a detailed account of related work, we refer to the bibliographic notes and the related work paragraphs of each chapter. In Section 8.2, we identify several issues that are left for future work.

8.1 Discussion

In this thesis, we are concerned with techniques, stemming from the area of Formal Methods. In the early phases of a software engineering process, techniques from Formal Methods are often employed to improve on and validate the design of a system. However, reality provides us with the evidence that such techniques are hardly, or not at all applied. Several reasons can be identified for this phenomenon. To start, applying such techniques are rather expensive: many systems can be (and in fact *are*) built without recourse to the techniques of Formal Methods. Thus, it is hard to show the added value of such techniques. Second, techniques from the area of Formal Methods are often considered difficult. Finally, as a last reason, it is often felt that the techniques that have been developed in the area of Formal Methods are applicable only for a very restricted class of problems, whereas the problems that could really benefit from some extra theory are left without the appropriate techniques.

Research in the area of Formal Methods focuses mainly on the latter two points of criticism. By developing new techniques that are applicable for ever larger classes of problems and are not unnecessarily complicated, we aim at furthering the state of the art in Formal Methods. It is the latter two points of criticism we are also concerned with in this thesis. In particular, we studied the following topics:

1. Verification and synthesis of finite and infinite state systems. In Chapter 3, we develop a technique for side-stepping the finite state-space requirement for the model-checking technique. The technique we discuss proves to be quite useful in the case of data-dependent systems; such systems are abundantly present in practice. We report on a prototype tool that implements our technique and show that it can deal with several realistic protocols. The techniques, developed in Chapter 4, allow us to restrict our analysis to a smaller region of the entire state space. Moreover, these techniques can lend a helping hand in developing

controllers for generic components that have to plugged into different kinds of environments.

2. Semantics of process languages for real-time and hybrid systems. In Chapter 5, we provide two types of semantics (one standard and one “non-standard”) for each of the two more popular modelling languages in computer science, viz. timed automata and hybrid automata. We show both types of semantics coincide for timed automata, and identify a mismatch between the two types of semantics we gave for hybrid automata. We then proceed to show this is due to the absence of a (generally accepted) property of the standard semantics. However, we also show that for a class of hybrid automata, viz. *approximable hybrid automata*, both types of semantics coincide.

   The results, obtained in Chapter 5 serve to illuminate the translations of timed automata and hybrid automata into μCRL$_t$, given in Chapter 6. These translations show that μCRL$_t$ is at least as expressive as timed automata (in fact, we also show it is strictly more expressive by considering a μCRL$_t$ expression that cannot be turned into a timed automaton). We furthermore show that for approximable hybrid automata, we can find corresponding μCRL$_t$ expressions, proving that μCRL$_t$ is at least as expressive as this particular class of hybrid automata. For non-approximable hybrid automata, both languages are incomparable.

3. Maturity of a μCRL$_t$. The claims in the defining documents of μCRL$_t$ suggest that it was designed in such a way that standard analysis techniques carry over rather straightforwardly to the timed setting. In Chapter 7, we show that we have in fact only several tools at our disposal to analyse timed (and hybrid) systems, and that some of these tools are not very easy to use. We show that, in order for μCRL$_t$ to be successful in the area of real-time and hybrid systems, the necessary tool support must be developed. Our conclusion is that μCRL$_t$ still is a far from mature language, and can be improved on in several directions.

The topics we studied suggest that parts of the criticism on Formal Methods are indeed justified. For instance, in Chapter 7, it turns out that the techniques that have been developed for μCRL$_t$ are difficult to apply. Moreover, several desired techniques for the analysis of real-time and hybrid systems have not even been developed for μCRL$_t$. The results in Chapter 5 and 6 reveal that not all languages for real-time and hybrid systems agree on basic concepts, such as *time additivity*. This shows that the theoretical framework is far from established. On a more positive note, we arrive at the conclusion that user-friendly techniques from Formal Methods, such as model-checking are reaching maturity and can be applied to real systems.

8.2 Future Work

Many issues still remain open. In fact, in this section, we probably hint at more questions than we have answered in this thesis. We address these chapter-wise.

The techniques, described in Chapter 3 have opened up the way for directly verifying μCRL processes without turning to verification techniques that depend on an explicit representation of the state space. However, the algorithm we provide in Chapter 3, does not always terminate in all cases. It is easy to see that undecidability of the modal μ-calculus for infinite state systems already follows from the halting problem: for any system, we can ask whether it eventually halts, which is expressed as $\mu X.[\mathrm{t}]X$. Note that this is a “regular” modal μ-calculus formula, rather than a first-order modal μ-calculus formula. This raises two questions:

1. for which class of systems can we guarantee termination of our algorithm?

2. are there (and if so: which) classes of logical formulae that are decidable, regardless of the processes they are verified on?

These questions are theoretical in nature, and answering them can be quite a challenge.

We next discuss two pragmatic attempts at increasing the success rate of our algorithm. In Chapter 3, we showed that we can employ techniques that in some cases allow us to eliminate quantifiers over an infinite domain. Finding novel techniques for quantifier elimination or reduction are necessary for increasing the success rate of the tool. Another promising direction is *pattern matching* for first-order boolean equation systems. Recall Example 3.3.4 in Chapter 3, describing a system that counts down from a randomly chosen natural number to zero and then randomly selects a new natural number (repeating the process *ad infinitum*). The associated equation system is $\mu Z(n{:}\mathbb{N}) = (n = 0 \vee Z(n-1))$. In this particular example, we showed our algorithm will not terminate when fed this equation system. However, it is straightforward to see that the smallest solution to this equation system is $Z(n{:}\mathbb{N}) = (\exists k{:}\mathbb{N}\ n - k = 0)$, which is true for any $n{:}\mathbb{N}$. The identification (and automatic recognition) of such patterns can lead to drastic improvements of our tool.

Several issues have remained unaddressed in Chapter 4. One of these issues is the pursuit of automation of the proposed techniques. Another issue is the identification of other classes of processes that are in some sense “easier” to prove strongly controllable. A follow up on this particular chapter is the study of the so-named *multi-step* control problem. Needless to say, this is at least as complex as the single-step control problem we addressed. Finally, one can think of extensions of the techniques we discussed into the area of real-time systems. This is bound to raise several philosophical questions: since we work in a setting of absolute timing, one can state properties like “time is not allowed to progress beyond point in time 10”. Such properties can induce phenomena such as Zeno behaviour. A questions that has to be answered is whether we allow a controller to “stop time” for a process, i.e. put a time-limit on the executions of a process. The implications of including time into this theory are very complex.

In Chapter 5, we introduce the notion of *approximable hybrid automata*, as a means to identify a class of hybrid automata for which the standard semantics and our alternative semantics of hybrid automata coincide. The definition of approximability of flow constraints is in fact a rather operational notion. Whether an alternative, easier formulation of the same concept exists, which is based on the syntax of a flow constraint, rather than on its solutions is still an open question which deserves attention. Related to this question is the identification of other classes of hybrid automata that are also approximable hybrid automata. Answers to this latter question can provide evidence of the (un)importance of the time-additivity constraint in the semantics of hybrid automata.

The translations of timed automata and hybrid automata we discuss in Chapter 6 call for tool support. This should not prove too hard to provide. In this chapter, we did not address a translation of (classes of) μCRL$_t$ expressions to timed automata. Such a translation would be more than welcome: using such a translation, we can employ established tools such as UPPAAL for model checking μCRL$_t$ expressions. Then, μCRL$_t$ would seemingly combine the best of both worlds: axiomatic reasoning on the level of μCRL$_t$, and automated model checking on the level of timed automata. An interesting observation in this direction is made by Fokkink (Chapter 6 of [44]), where he provides a sketch of a translation of an expression in ACP with prefix integration to timed automata without invariants. Given the similarities between μCRL and ACP, it is well conceivable that his translation can be modified to work for μCRL$_t$.

Finally, in Chapter 7, we have identified several gaps in the applicability of the theory of μCRL$_t$. For instance, the encapsulation and elimination theorems that appear in [57], are not useful in practice, due to the incredible amount of time-locks that are generated. We have the feeling that there is some room for improvement, since this blow-up does not seem to occur when using different versions of real-time ACP. Finally, the tools for performing some kind of performance analysis within the theory of μCRL$_t$ are completely absent and thus need to be developed. Such tools can range from automation for the encapsulation and elimination theorems to simulators for μCRL$_t$ expressions to new equivalence relations that are coarser than timed (branching) bisimulation.

Appendix A

The terms resulting from encapsulating and expanding the belt system $S(\overline{0})$ are listed in the below table. Only the terms clearly satisfying the invariants, listed in Table 7.10 are shown. The first collumn lists the source alternatives, yielding the term, listed in the second column. The third column annotates the time-lock terms with the (combination of) source pairs that allow us to establish that the time-lock does not occur. We use a "C" preceding the combination of source pairs yielding a communication and an "A" preceding the combination of source pairs yielding the autonomous $\bar{c}$ action. If the condition of a time-lock can be reduced to false, using the invariants, we use the annotation f.

Pair	Term	Subterm of
(B1+C2)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \mathsf{a}\text{‹}(u+d)\cdot$ $S(u+d, u+d, \Delta, 1, 3, b\mu(u-t+d)+s_b, 0, y_b, b\mu d+s_c, 0, \mu)$ $\lhd \varsigma = 1 \land s_c \geq 0 \land 0 \leq s_c+b\mu d \leq v_{\mathit{max}} \land s_b+b\mu(u-t+d) \geq 0 \land \sigma = 0 \rhd \delta\text{‹}\mathbf{0}+$	
(B1+C4)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \mathsf{a}\text{‹}(u+d)\cdot$ $S(u+d, u+d, \Delta, 1, 4, b\mu(u-t+d)+s_b, 0, y_b, b\mu d+s_c, 0, \mu)$ $\lhd \varsigma = 2 \land 0 \leq s_c+b\mu d \leq v_{\mathit{max}} \land d \leq \Delta \land s_b+b\mu(u-t+d) \geq 0 \land \sigma = 0 \rhd \delta\text{‹}\mathbf{0}+$	
(B2+C3)	$\mathrm{F}(0)\text{‹}(u - \frac{1}{b\mu}s_c)\cdot$ $S(u - \frac{1}{b\mu}s_c, u - \frac{1}{b\mu}s_c, \Delta, 0, 1, b\mu(u-t) + s_b - s_c, x_b, y_b, 0, x_c, 0)$ $\lhd \varsigma = 1 \land b\mu \neq 0 \land \sigma = 0 \rhd \delta\text{‹}\mathbf{0}+$	
(B2+C5)	$\mathrm{F}(0)\text{‹}(u + \frac{1}{b\mu}(v_{\mathit{max}} - s_c))\cdot$ $S(u + \frac{1}{b\mu}(v_{\mathit{max}} - s_c), u + \frac{1}{b\mu}(v_{\mathit{max}} - s_c), \Delta - \frac{1}{b\mu}(v_{\mathit{max}} - s_c), 0, 2, b\mu(u-t) + v_{\mathit{max}} - s_c + s_b, x_b, y_b, v_{\mathit{max}}, x_c, 0) \lhd \varsigma = 2 \land b\mu \neq 0 \land v_{\mathit{max}} - s_c \leq \Delta b\mu \land \sigma = 0 \rhd \delta\text{‹}\mathbf{0}+$	
(B2+C6)	$\mathrm{F}(-T_{\mathit{max}})\text{‹}(u+\Delta)\cdot$ $S(u+\Delta, u+\Delta, \Delta, 0, 1, b\mu(u-t+\Delta) + s_b, x_b, y_b, b\mu\Delta + s_c, x_c, -T_{\mathit{max}})$ $\lhd \varsigma = 2 \land 0 \leq s_c + b\mu\Delta \leq v_{\mathit{max}} \land \sigma = 0 \rhd \delta\text{‹}\mathbf{0}+$	
(B3+C12)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \mathsf{a}\text{‹}(u+d)\cdot$	

Terms resulting from encapsulating and expanding

Pair	Term	Subterm of
	$S(u+d,u+d,\Delta,2,6,b\mu(u-t+d)+s_b,0,\frac{1}{2}b\mu(u-t+d)^2+s_b(u-t+d)+x_b,s_c+b\mu d,0,\mu) \lhd s_b+b\mu(u-t+d)\geq 0\wedge\varsigma=5\wedge 0\leq s_c+b\mu d\leq v_{max}\wedge\sigma=1\wedge 0\leq\frac{1}{2}b\mu(u-t+d)^2+s_b(u-t+d)+x_b\leq l\rhd\delta{}^{c}\mathbf{0}+$	
(B6+C13)	$\mathrm{d}{}^{c}(t+\frac{1}{b\mu}(-s_b+\sqrt{s_b^2-2b\mu(x_b-l)}))\cdot S(t+\frac{1}{b\mu}(-s_b+\sqrt{s_b^2-2b\mu(x_b-l)}),t+\frac{1}{b\mu}(-s_b+\sqrt{s_b^2-2b\mu(x_b-l)}),\mathfrak{d},0,2,\sqrt{s_b^2-2b\mu(x_b-l)},x_b,y_b,s_c-s_b+\sqrt{s_b^2-2b\mu(x_b-l)}+b\mu(t-u),x_c,\mu)\lhd b\mu\neq 0\wedge\sqrt{s_b^2-2b\mu(x_b-l)}\geq 0\wedge\varsigma=5\wedge 0\leq s_c-s_b+\sqrt{s_b^2-2b\mu(x_b-l)}+b\mu(t-u)\leq v_{max}\wedge\sigma=1\rhd\delta{}^{c}\mathbf{0}+$	
(B7+C13)	$\mathrm{d}{}^{c}(t+\frac{l-x_b}{s_b})\cdot S(t+\frac{l-x_b}{s_b},t+\frac{l-x_b}{s_b},\mathfrak{d},0,2,s_b,x_b,y_b,s_c,x_c,\mu)\lhd b\mu=0\wedge s_b\geq 0\wedge\varsigma=5\wedge 0\leq s_c\leq v_{max}\wedge\sigma=1\rhd\delta{}^{c}\mathbf{0}+$	
(B8+C7)	$\mathrm{F}(0){}^{c}u\cdot S(u,u,\Delta,1,4,s_b+b\mu(u-t),\frac{1}{2}b\mu(u-t)^2+s_b(u-t)+x_b,y_b,s_c,x_c,0)\lhd 0\leq\frac{1}{2}b\mu(u-t)^2+s_b(u-t)+x_b\leq l\wedge\varsigma=3\wedge s_c=v_{max}\wedge\sigma=1\rhd\delta{}^{c}\mathbf{0}+$	
(B8+C8)	$\mathrm{F}(T_{max}){}^{c}u\cdot S(u,u,\Delta,1,4,s_b+b\mu(u-t),\frac{1}{2}b\mu(u-t)^2+s_b(u-t)+x_b,y_b,s_c,x_c,T_{max})\lhd 0\leq\frac{1}{2}b\mu(u-t)^2+s_b(u-t)+x_b\leq l\wedge\varsigma=3\wedge s_c\neq v_{max}\wedge\sigma=1\rhd\delta{}^{c}\mathbf{0}+$	
(B8+C9)	$\mathrm{F}(0){}^{c}(u+\frac{v_{max}-s_c}{b\mu})\cdot S(u+\frac{v_{max}-s_c}{b\mu},u+\frac{v_{max}-s_c}{b\mu},\Delta,1,4,s_b-s_c+v_{max}+b\mu(u-t),\frac{1}{2}b\mu(u-t+\frac{v_{max}-s_c}{b\mu})^2+s_b(u-t+\frac{v_{max}-s_c}{b\mu})+x_b,y_b,v_{max},\frac{1}{2}b\mu(\frac{v_{max}-s_c}{b\mu})^2+s_c\frac{v_{max}-s_c}{b\mu}+x_c,0)\lhd 0\leq\frac{1}{2}b\mu(u-t+\frac{v_{max}-s_c}{b\mu})^2+s_b(u-t+\frac{v_{max}-s_c}{b\mu})+x_b\leq l\wedge\varsigma=4\wedge 0\leq\frac{1}{2}b\mu(\frac{v_{max}-s_c}{b\mu})^2+s_c\frac{v_{max}-s_c}{b\mu}+x_c\leq\frac{1}{2}l\wedge b\mu\neq 0\wedge\sigma=1\rhd\delta{}^{c}\mathbf{0}+$	
(B8+C14)	$\mathrm{F}(0){}^{c}(u+\frac{v_{max}-s_c}{b\mu})\cdot S(u+\frac{v_{max}-s_c}{b\mu},u+\frac{v_{max}-s_c}{b\mu},\Delta,1,5,s_b-s_c+v_{max}+b\mu(u-t),\frac{1}{2}b\mu(u-t+\frac{v_{max}-s_c}{b\mu})^2+s_b(u-t+\frac{v_{max}-s_c}{b\mu})+x_b,y_b,v_{max},x_c+\frac{1}{2}b\mu(\frac{v_{max}-s_c}{b\mu})^2+s_c\frac{v_{max}-s_c}{b\mu},0)\lhd 0\leq\frac{1}{2}b\mu(u-t+\frac{v_{max}-s_c}{b\mu})^2+s_b(u-t+\frac{v_{max}-s_c}{b\mu})+x_b\leq l\wedge\varsigma=5\wedge b\mu\neq 0\wedge\sigma=1\rhd\delta{}^{c}\mathbf{0}+$	
(B12+C15)	$\mathrm{d}{}^{c}(t+\frac{-s_b+\sqrt{s_b^2-2b\mu(y_b-l)}}{b\mu})\cdot$	

Terms resulting from encapsulating and expanding

Pair	Term	Subterm of
	$S(t + \frac{1}{b\mu}(-s_b + \sqrt{s_b^2 - 2b\mu(y_b - l)}), t + \frac{1}{b\mu}(-s_b + \sqrt{s_b^2 - 2b\mu(y_b - l)}), \Delta, 1, 4, \sqrt{s_b^2 - 2b\mu(y_b - l)}, x_b + (l - y_b), y_b, s_c - s_b + \sqrt{s_b^2 - 2b\mu(y_b - l)} + b\mu(t-u), \frac{1}{2}b\mu(t-u+\frac{-s_b+\sqrt{s_b^2-2b\mu(y_b-l)}}{b\mu})^2 + s_c(t-u+\frac{-s_b+\sqrt{s_b^2-2b\mu(y_b-l)}}{b\mu}) + x_c, \mu)$ $\triangleleft b\mu \neq 0 \wedge \sqrt{s_b^2 - 2b\mu(y_b - l)} \geq 0 \wedge \varsigma = 6 \wedge 0 \leq s_c - s_b + \sqrt{s_b^2 - 2b\mu(y_b - l)} + b\mu(t - u) \leq v_{max} \wedge \sigma = 2 \triangleright \delta\text{‹}\mathbf{0}+$	
(B13+C15)	$\mathrm{d}\text{‹}(t + \frac{l-y_b}{s_b})\cdot$ $S(t+\frac{l-y_b}{s_b}, t+\frac{l-y_b}{s_b}, \Delta, 1, 4, s_b, x_b + (l - y_b), y_b, s_c + b\mu(t - u + \frac{l-y_b}{s_b}), s_c(t - u + \frac{l-y_b}{s_b}) + x_c, \mu)$ $\triangleleft b\mu = 0 \wedge s_b \geq 0 \wedge \varsigma = 6 \wedge 0 \leq s_c + b\mu(t - u + \frac{l-y_b}{s_b}) \leq v_{max} \wedge \sigma = 2 \triangleright \delta\text{‹}\mathbf{0}+$	
(B14+C16)	$\mathrm{F}(0)\text{‹}(u + \frac{v_{max}-s_c}{b\mu})\cdot$ $S(u + \frac{v_{max}-s_c}{b\mu}, u + \frac{v_{max}-s_c}{b\mu}, \Delta, 2, 6, s_b - s_c + v_{max} + b\mu(u - t), \frac{1}{2}b\mu(u-t+\frac{v_{max}-s_c}{b\mu})^2 + s_b(u-t+\frac{v_{max}-s_c}{b\mu}) + x_b, \frac{1}{2}b\mu(u-t+\frac{v_{max}-s_c}{b\mu})^2 + s_b(u-t+\frac{v_{max}-s_c}{b\mu}) + y_b, v_{max}, \frac{1}{2}b\mu(\frac{v_{max}-s_c}{b\mu})^2 + s_c\frac{v_{max}-s_c}{b\mu} + x_c, 0)$ $\triangleleft 0 \leq \frac{1}{2}b\mu(u - t + \frac{v_{max}-s_c}{b\mu})^2 + s_b(u - t + \frac{v_{max}-s_c}{b\mu}) + x_b \leq \frac{1}{2}b\mu(u - t + \frac{v_{max}-s_c}{b\mu})^2 + s_b(u - t + \frac{v_{max}-s_c}{b\mu}) + y_b \leq l \wedge \varsigma = 6 \wedge b\mu \neq 0 \wedge \sigma = 2 \triangleright \delta\text{‹}\mathbf{0}$	
(B1 +C1)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \overline{c}\text{‹}0\cdot$ $S(t, 0, 0, \sigma, 1, s_b, x_b, y_b, 0, 0, \mu)$ $\triangleleft\varsigma = 0 \wedge 0 \leq t + d \wedge s_b + b\mu d \geq 0 \wedge \sigma = 0 \triangleright \delta\text{‹}\mathbf{0}+$	
(B2+C1)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \overline{c}\text{‹}0\cdot$ $S(t, 0, 0, \sigma, 1, s_b, x_b, y_b, 0, 0, \mu)$ $\triangleleft\varsigma = 0 \wedge 0 \leq t + d \wedge \sigma = 0 \triangleright \delta\text{‹}\mathbf{0}+$	
(B3+C10)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \overline{c}\text{‹}(u + \frac{1}{b\mu}(-s_c + \sqrt{s_c^2 - 2b\mu(x_c - \frac{1}{2}l)}))\cdot$ $S(t, u + \frac{1}{b\mu}(-s_c + \sqrt{s_c^2 - 2b\mu(x_c - \frac{1}{2}l)}), \Delta, \sigma, 5, s_b, x_b, y_b, \sqrt{s_c^2 - b\mu(2x_c - l)}, \frac{1}{2}l, \mu)$ $\triangleleft\varsigma = 4 \wedge u + \frac{-s_c+\sqrt{s_c^2-2b\mu(x_c-\frac{1}{2}l)}}{b\mu} \leq t + d \wedge s_b + b\mu d \geq 0 \wedge \sqrt{s_c^2 - b\mu(2x_c - l)} \leq v_{max} \wedge b\mu \neq 0 \wedge 0 \leq \frac{1}{2}b\mu d^2 + s_b d + x_b \leq l \wedge \sigma = 1 \triangleright \delta\text{‹}\mathbf{0}+$	
(B6+C10)	$\overline{c}\text{‹}(u + \frac{1}{b\mu}(-s_c + \sqrt{s_c^2 - 2b\mu(x_c - \frac{1}{2}l)}))\cdot$ $S(t, u + \frac{1}{b\mu}(-s_c + \sqrt{s_c^2 - 2b\mu(x_c - \frac{1}{2}l)}), \Delta, \sigma, 5, s_b, x_b, y_b, \sqrt{s_c^2 - b\mu(2x_c - l)}, \frac{1}{2}l, \mu)$	

Terms resulting from encapsulating and expanding

Pair	Term	Subterm of
	$\lhd \varsigma = 4 \wedge u + \frac{-s_c+\sqrt{s_c^2-2b\mu(x_c-\frac{1}{2}l)}}{b\mu} \leq t + \frac{-s_b+\sqrt{s_b^2-2b\mu(x_b-l)}}{b\mu} \wedge$ $b\mu \neq 0 \wedge \sqrt{s_b^2 - 2b\mu(x_b - l)} \geq 0 \wedge \sqrt{s_c^2 - b\mu(2x_c - l)} \leq$ $v_{max} \wedge \sigma = 1 \rhd \delta\text{‹}\mathbf{0}+$	
(B8+C10)	$\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \overline{c}\text{‹}(u + \frac{1}{b\mu}(-s_c + \sqrt{s_c^2 - 2b\mu(x_c - \frac{1}{2}l)}))\cdot$ $S(t, u + \frac{1}{b\mu}(-s_c + \sqrt{s_c^2 - 2b\mu(x_c - \frac{1}{2}l)}), \Delta, \sigma, 5, s_b, x_b, y_b,$ $\sqrt{s_c^2 - b\mu(2x_c - l)}, \frac{1}{2}l, \mu)$ $\lhd \varsigma = 4 \wedge u + \frac{-s_c+\sqrt{s_c^2-2b\mu(x_c-\frac{1}{2}l)}}{b\mu} \leq t + d \wedge 0 \leq \frac{1}{2}b\mu d^2 + s_b d +$ $x_b \leq l \wedge \sqrt{s_c^2 - b\mu(2x_c - l)} \leq v_{max} \wedge b\mu \neq 0 \wedge \sigma = 1 \rhd \delta\text{‹}\mathbf{0}+$	
(B3+C11)	$\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \overline{c}\text{‹}(u + \frac{\frac{1}{2}l-x_c}{s_c})\cdot$ $S(t, u + \frac{\frac{1}{2}l-x_c}{s_c}, \Delta, \sigma, 5, s_b, x_b, y_b, v_{max}, \frac{1}{2}l, \mu)$ $\lhd \varsigma = 4 \wedge u + \frac{\frac{1}{2}l-x_c}{s_c} \leq t + d \wedge s_b + b\mu d \geq 0 \wedge b\mu = 0 \wedge \sigma =$ $1 \wedge 0 \leq \frac{1}{2}b\mu d^2 + s_b d + x_b \leq l \rhd \delta\text{‹}\mathbf{0}+$	
(B7+C11)	$\overline{c}\text{‹}(u + \frac{\frac{1}{2}l-x_c}{s_c})\cdot$ $S(t, u + \frac{\frac{1}{2}l-x_c}{s_c}, \Delta, \sigma, 5, s_b, x_b, y_b, v_{max}, \frac{1}{2}l, \mu)$ $\lhd \varsigma = 4 \wedge u + \frac{\frac{1}{2}l-x_c}{s_c} \leq t + \frac{l-x_b}{s_b} \wedge s_b \geq 0 \wedge b\mu = 0 \wedge \sigma = 1 \rhd \delta\text{‹}\mathbf{0}+$	
(B8+C11)	$\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \overline{c}\text{‹}(u + \frac{\frac{1}{2}l-x_c}{s_c})\cdot$ $S(t, u + \frac{\frac{1}{2}l-x_c}{s_c}, \Delta, \sigma, 5, s_b, x_b, y_b, v_{max}, \frac{1}{2}l, \mu)$ $\lhd \varsigma = 4 \wedge u + \frac{\frac{1}{2}l-x_c}{s_c} \leq t + d \wedge 0 \leq \frac{1}{2}b\mu d^2 + s_b d + x_b \leq$ $l \wedge b\mu = 0 \wedge \sigma = 1 \rhd \delta\text{‹}\mathbf{0}+$	
(B1+C1)	$\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \delta\text{‹}(t + d)$ $\lhd t + d \leq 0 \wedge \varsigma = 0 \wedge s_b + b\mu d \geq 0 \rhd \delta\text{‹}\mathbf{0}+$	A(B1+C1)
(B1+C2)	$\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \sum_{d':\mathbb{R}_{\geq \mathbf{0}}} \delta\text{‹}(t + d)$ $\lhd t + d \leq u + d' \wedge \varsigma = 1 \wedge s_b + b\mu d \geq 0 \wedge s_c \geq 0 \wedge 0 \leq$ $s_c + b\mu d' \leq v_{max} \rhd \delta\text{‹}\mathbf{0}+$	C(B1 + C2)
(B1+C3)	$\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \delta\text{‹}(t + d)$ $\lhd t + d \leq u + \frac{-s_c}{b\mu} \wedge \varsigma = 1 \wedge s_b + b\mu d \geq 0 \wedge b\mu \neq 0 \rhd \delta\text{‹}\mathbf{0}+$	C(B2 + C3)
(B1+C4)	$\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \sum_{d':\mathbb{R}_{\geq \mathbf{0}}} \delta\text{‹}(t + d)$ $\lhd t + d \leq u + d' \wedge \varsigma = 2 \wedge s_b + b\mu d \geq 0 \wedge 0 \leq s_c + b\mu d' \leq$ $v_{max} \wedge d' \leq \Delta \rhd \delta\text{‹}\mathbf{0}+$	C(B1+C4)
(B1+C5)	$\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \delta\text{‹}(t + d)$ $\lhd t + d \leq u + \frac{v_{max}-s_c}{b\mu} \wedge \varsigma = 2 \wedge s_b + b\mu d \geq 0 \wedge b\mu \neq$ $0 \wedge \frac{v_{max}-s_c}{b\mu} \leq \Delta \rhd \delta\text{‹}\mathbf{0}+$	C(B2+C5)
(B1+C6)	$\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \delta\text{‹}(t + d)$	C(B2+C6)

Terms resulting from encapsulating and expanding

Pair	Term	Subterm of
	$\lhd t+d \leq u+\Delta \wedge \varsigma = 2 \wedge s_b + b\mu d \geq 0 \wedge 0 \leq s_c + b\mu\Delta \leq v_{\mathit{max}} \rhd \delta\text{`}\mathbf{0} +$	
(B2+C1)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \delta\text{`}(t+d)$ $\lhd t+d \leq 0 \wedge \varsigma = 0 \rhd \delta\text{`}\mathbf{0} +$	A(B2 +C1)
(B2+C2)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \sum_{d':\mathbb{R}_{\geq\mathbf{0}}} \delta\text{`}(t+d)$ $\lhd t+d \leq u+d' \wedge \varsigma = 1 \wedge s_c \geq 0 \wedge 0 \leq s_c + b\mu d' \leq v_{\mathit{max}} \rhd \delta\text{`}\mathbf{0} +$	C(B1+C2)
(B2+C3)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \delta\text{`}(t+d)$ $\lhd t+d \leq u + \frac{-s_c}{b\mu} \wedge \varsigma = 1 \wedge b\mu \neq 0 \rhd \delta\text{`}\mathbf{0} +$	C(B2+C3)
(B2+C4)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \sum_{d':\mathbb{R}_{\geq\mathbf{0}}} \delta\text{`}(t+d)$ $\lhd t+d \leq u+d' \wedge \varsigma = 2 \wedge 0 \leq s_c + b\mu d' \leq v_{\mathit{max}} \wedge d' \leq \Delta \rhd \delta\text{`}\mathbf{0} +$	C(B2+C6)
(B2+C5)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \delta\text{`}(t+d)$ $\lhd t+d \leq u + \frac{v_{\mathit{max}} - s_c}{b\mu} \wedge \varsigma = 2 \wedge b\mu \neq 0 \wedge \frac{v_{\mathit{max}} - s_c}{b\mu} \leq \Delta \rhd \delta\text{`}\mathbf{0} +$	C(B2+C5)
(B2+C6)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \delta\text{`}(t+d)$ $\lhd t+d \leq u+\Delta \wedge \varsigma = 2 \wedge 0 \leq s_c + b\mu\Delta \leq v_{\mathit{max}} \rhd \delta\text{`}\mathbf{0} +$	C(B2+C6)
(B3+C7)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \delta\text{`}(t+d)$ $\lhd t+d \leq u \wedge \varsigma = 3 \wedge s_b + b\mu d \geq 0 \wedge s_c = v_{\mathit{max}} \wedge 0 \leq \frac{1}{2}b\mu d^2 + s_b d + x_b \leq l \rhd \delta\text{`}\mathbf{0} +$	C(B8+C7)
(B3+C8)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \delta\text{`}(t+d)$ $\lhd t+d \leq u \wedge \varsigma = 3 \wedge s_b + b\mu d \geq 0 \wedge s_c \neq v_{\mathit{max}} \wedge 0 \leq \frac{1}{2}b\mu d^2 + s_b d + x_b \leq l \rhd \delta\text{`}\mathbf{0} +$	C(B8 +C8)
(B3+C9)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \delta\text{`}(t+d)$ $\lhd t+d \leq u + \frac{v_{\mathit{max}} - s_c}{b\mu} \wedge \varsigma = 4 \wedge s_b + b\mu d \geq 0 \wedge \frac{1}{2}b\mu(\frac{v_{\mathit{max}} - s_c}{b\mu})^2 + s_c \frac{v_{\mathit{max}} - s_c}{b\mu} + x_c \leq \frac{1}{2}l \wedge b\mu \neq 0 \wedge 0 \leq \frac{1}{2}b\mu d^2 + s_b d + x_b \leq l \rhd \delta\text{`}\mathbf{0} +$	C(B8 +C9)
(B3+C10)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \delta\text{`}(t+d)$ $\lhd t+d \leq u + \frac{-s_c + \sqrt{s_c^2 - 2b\mu(x_c - \frac{1}{2}l)}}{b\mu} \wedge \varsigma = 4 \wedge s_b + b\mu d \geq 0 \wedge \sqrt{s_c^2 - b\mu(2x_c - l)} \leq v_{\mathit{max}} \wedge b\mu \neq 0 \wedge 0 \leq \frac{1}{2}b\mu d^2 + s_b d + x_b \leq l \rhd \delta\text{`}\mathbf{0} +$	A(B3+C10)
(B3+C11)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \delta\text{`}(t+d)$ $\lhd t+d \leq u + \frac{\frac{1}{2}l - x_c}{s_c} \wedge \varsigma = 4 \wedge s_b + b\mu d \geq 0 \wedge b\mu = 0 \wedge 0 \leq \frac{1}{2}b\mu d^2 + s_b d + x_b \leq l \rhd \delta\text{`}\mathbf{0} +$	A(B3+C11)
(B3+C12)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \sum_{d':\mathbb{R}_{\geq\mathbf{0}}} \delta\text{`}(t+d)$ $\lhd t+d \leq u+d' \wedge \varsigma = 5 \wedge s_b + b\mu d \geq 0 \wedge 0 \leq s_c + b\mu d' \leq v_{\mathit{max}} \wedge 0 \leq \frac{1}{2}b\mu d^2 + s_b d + x_b \leq l \rhd \delta\text{`}\mathbf{0} +$	C(B3+C12)
(B3+C13)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \sum_{d':\mathbb{R}_{\geq\mathbf{0}}} \delta\text{`}(t+d)$ $\lhd t+d \leq u+d' \wedge \varsigma = 5 \wedge s_b + b\mu d \geq 0 \wedge 0 \leq s_c + b\mu d' \leq v_{\mathit{max}} \wedge 0 \leq \frac{1}{2}b\mu d^2 + s_b d + x_b \leq l \rhd \delta\text{`}\mathbf{0} +$	C(B3+C12)
(B3+C14)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \delta\text{`}(t+d)$	C(B8 + C7)

Terms resulting from encapsulating and expanding

Pair	Term	Subterm of
	$\lhd t + d \leq u + \frac{v_{max} - s_c}{b\mu} \land \varsigma = 5 \land s_b + b\mu d \geq 0 \land b\mu \neq 0 \land 0 \leq \frac{1}{2}b\mu d^2 + s_b d + x_b \leq l \rhd \delta\text{‘}\mathbf{0} +$	
(B4+C7)	$\delta\text{‘}(t + \frac{-s_b - \sqrt{s_b^2 - 2b\mu x_b}}{b\mu})$ $\lhd t + \frac{-s_b - \sqrt{s_b^2 - 2b\mu x_b}}{b\mu} \leq u \land \varsigma = 3 \land b\mu \neq 0 \land \sqrt{s_b^2 - 2b\mu x_b} > 0 \land s_c = v_{max} \rhd \delta\text{‘}\mathbf{0} +$	C(B8+C7)
(B4+C8)	$\delta\text{‘}(t + \frac{-s_b - \sqrt{s_b^2 - 2b\mu x_b}}{b\mu})$ $\lhd t + \frac{-s_b - \sqrt{s_b^2 - 2b\mu x_b}}{b\mu} \leq u \land \varsigma = 3 \land b\mu \neq 0 \land \sqrt{s_b^2 - 2b\mu x_b} > 0 \land s_c \neq v_{max} \rhd \delta\text{‘}\mathbf{0} +$	C(B8+C8)
(B4+C9)	$\delta\text{‘}(t + \frac{-s_b - \sqrt{s_b^2 - 2b\mu x_b}}{b\mu})$ $\lhd t + \frac{-s_b - \sqrt{s_b^2 - 2b\mu x_b}}{b\mu} \leq u + \frac{v_{max} - s_c}{b\mu} \land \varsigma = 4 \land b\mu \neq 0 \land \sqrt{s_b^2 - 2b\mu x_b} > 0 \land \frac{1}{2}b\mu(\frac{v_{max} - s_c}{b\mu})^2 + s_c\frac{v_{max} - s_c}{b\mu} + x_c \leq \frac{1}{2}l \rhd \delta\text{‘}\mathbf{0} +$	C(B8+C9)
(B4+C10)	$\delta\text{‘}(t + \frac{-s_b - \sqrt{s_b^2 - 2b\mu x_b}}{b\mu})$ $\lhd t + \frac{-s_b - \sqrt{s_b^2 - 2b\mu x_b}}{b\mu} \leq u + \frac{-s_c + \sqrt{s_c^2 - 2b\mu(x_c - \frac{1}{2}l)}}{b\mu} \land \varsigma = 4 \land b\mu \neq 0 \land \sqrt{s_b^2 - 2b\mu x_b} > 0 \land \sqrt{s_c^2 - b\mu(2x_c - l)} \leq v_{max} \rhd \delta\text{‘}\mathbf{0} +$	C(B8+C9) A(C10)
(B4+C12)	$\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \delta\text{‘}(t + \frac{-s_b - \sqrt{s_b^2 - 2b\mu x_b}}{b\mu})$ $\lhd t + \frac{-s_b - \sqrt{s_b^2 - 2b\mu x_b}}{b\mu} \leq u + d \land \varsigma = 5 \land b\mu \neq 0 \land \sqrt{s_b^2 - 2b\mu x_b} > 0 \land 0 \leq s_c + b\mu d \leq v_{max} \rhd \delta\text{‘}\mathbf{0} +$	C(B3+C12)
(B4+C13)	$\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \delta\text{‘}(t + \frac{-s_b - \sqrt{s_b^2 - 2b\mu x_b}}{b\mu})$ $\lhd t + \frac{-s_b - \sqrt{s_b^2 - 2b\mu x_b}}{b\mu} \leq u + d \land \varsigma = 5 \land b\mu \neq 0 \land \sqrt{s_b^2 - 2b\mu x_b} > 0 \land 0 \leq s_c + b\mu d \leq v_{max} \rhd \delta\text{‘}\mathbf{0} +$	C(B3+C12)
(B4+C14)	$\delta\text{‘}(t + \frac{-s_b - \sqrt{s_b^2 - 2b\mu x_b}}{b\mu})$ $\lhd t + \frac{-s_b - \sqrt{s_b^2 - 2b\mu x_b}}{b\mu} \leq u + \frac{v_{max} - s_c}{b\mu} \land \varsigma = 5 \land b\mu \neq 0 \land \sqrt{s_b^2 - 2b\mu x_b} > 0 \rhd \delta\text{‘}\mathbf{0} +$	C(B3+C12)
(B6+C7)	$\delta\text{‘}(t + \frac{-s_b + \sqrt{s_b^2 - 2b\mu(x_b - l)}}{b\mu})$ $\lhd t + \frac{-s_b + \sqrt{s_b^2 - 2b\mu(x_b - l)}}{b\mu} \leq u \land \varsigma = 3 \land b\mu \neq 0 \land \sqrt{s_b^2 - 2b\mu(x_b - l)} \geq 0 \land s_c = v_{max} \rhd \delta\text{‘}\mathbf{0} +$	C(B8+C7)
(B6+C8)	$\delta\text{‘}(t + \frac{-s_b + \sqrt{s_b^2 - 2b\mu(x_b - l)}}{b\mu})$ $\lhd t + \frac{-s_b + \sqrt{s_b^2 - 2b\mu(x_b - l)}}{b\mu} \leq u \land \varsigma = 3 \land b\mu \neq 0 \land \sqrt{s_b^2 - 2b\mu(x_b - l)} \geq 0 \land s_c \neq v_{max} \rhd \delta\text{‘}\mathbf{0} +$	C(B8+C8)

Terms resulting from encapsulating and expanding

Pair	Term	Subterm of
(B6+C9)	$\delta\,{}^{c}(t + \frac{-s_b+\sqrt{s_b^2-2b\mu(x_b-l)}}{b\mu}) \lhd t + \frac{-s_b+\sqrt{s_b^2-2b\mu(x_b-l)}}{b\mu} \leq u + \frac{v_{max}-s_c}{b\mu} \wedge \varsigma = 4 \wedge b\mu \neq 0 \wedge \sqrt{s_b^2 - 2b\mu(x_b - l)} \geq 0 \wedge \frac{1}{2}b\mu(\frac{v_{max}-s_c}{b\mu})^2 + s_c\frac{v_{max}-s_c}{b\mu} + x_c \leq \frac{1}{2}l \rhd \delta\,{}^{c}\mathbf{0}+$	C(B8+C9)
(B6+C10)	$\delta\,{}^{c}(t + \frac{-s_b+\sqrt{s_b^2-2b\mu(x_b-l)}}{b\mu}) \lhd t + \frac{-s_b+\sqrt{s_b^2-2b\mu(x_b-l)}}{b\mu} \leq u + \frac{-s_c+\sqrt{s_c^2-2b\mu(x_c-\frac{1}{2}l)}}{b\mu} \wedge \varsigma = 4 \wedge b\mu \neq 0 \wedge \sqrt{s_b^2 - 2b\mu(x_b - l)} \geq 0 \wedge \sqrt{s_c^2 - b\mu(2x_c - l)} \leq v_{max} \rhd \delta\,{}^{c}\mathbf{0}+$	A(C10)

(B6+C12)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \delta\,{}^{c}(t + \frac{-s_b+\sqrt{s_b^2-2b\mu(x_b-l)}}{b\mu}) \lhd t + \frac{-s_b+\sqrt{s_b^2-2b\mu(x_b-l)}}{b\mu} \leq u + d \wedge \varsigma = 5 \wedge b\mu \neq 0 \wedge \sqrt{s_b^2 - 2b\mu(x_b - l)} \geq 0 \wedge 0 \leq s_c + b\mu d \leq v_{max} \rhd \delta\,{}^{c}\mathbf{0}+$	C(B3+C12)
		C(B6+C13)

(B6+C13)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \delta\,{}^{c}(t + \frac{-s_b+\sqrt{s_b^2-2b\mu(x_b-l)}}{b\mu}) \lhd t + \frac{-s_b+\sqrt{s_b^2-2b\mu(x_b-l)}}{b\mu} \leq u + d \wedge \varsigma = 5 \wedge b\mu \neq 0 \wedge \sqrt{s_b^2 - 2b\mu(x_b - l)} \geq 0 \wedge 0 \leq s_c + b\mu d \leq v_{max} \rhd \delta\,{}^{c}\mathbf{0}+$	C(B6+C13)

(B6+C14)	$\delta\,{}^{c}(t + \frac{-s_b+\sqrt{s_b^2-2b\mu(x_b-l)}}{b\mu}) \lhd t + \frac{-s_b+\sqrt{s_b^2-2b\mu(x_b-l)}}{b\mu} \leq u + \frac{v_{max}-s_c}{b\mu} \wedge \varsigma = 5 \wedge b\mu \neq 0 \wedge \sqrt{s_b^2 - 2b\mu(x_b - l)} \geq 0 \rhd \delta\,{}^{c}\mathbf{0}+$	C(B3+C12)
		C(B6+C13)
		C(B8+C14)

(B7+C7)	$\delta\,{}^{c}(t + \frac{l-x_b}{s_b}) \lhd t + \frac{l-x_b}{s_b} \leq u \wedge \varsigma = 3 \wedge b\mu = 0 \wedge s_b \geq 0 \wedge s_c = v_{max} \rhd \delta\,{}^{c}\mathbf{0}+$	C(B8+C7)
(B7+C8)	$\delta\,{}^{c}(t + \frac{l-x_b}{s_b}) \lhd t + \frac{l-x_b}{s_b} \leq u \wedge \varsigma = 3 \wedge b\mu = 0 \wedge s_b \geq 0 \wedge s_c \neq v_{max} \rhd \delta\,{}^{c}\mathbf{0}+$	C(B8+C8)
(B7+C11)	$\delta\,{}^{c}(t + \frac{l-x_b}{s_b}) \lhd t + \frac{l-x_b}{s_b} \leq u + \frac{\frac{1}{2}l-x_c}{s_c} \wedge \varsigma = 4 \wedge b\mu = 0 \wedge s_b \geq 0 \rhd \delta\,{}^{c}\mathbf{0}+$	A(C11)
(B7+C12)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \delta\,{}^{c}(t + \frac{l-x_b}{s_b}) \lhd t + \frac{l-x_b}{s_b} \leq u + d \wedge \varsigma = 5 \wedge b\mu = 0 \wedge s_b \geq 0 \wedge 0 \leq s_c + b\mu d \leq v_{max} \rhd \delta\,{}^{c}\mathbf{0}+$	C(B7+C13)
(B7+C13)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \delta\,{}^{c}(t + \frac{l-x_b}{s_b}) \lhd t + \frac{l-x_b}{s_b} \leq u + d \wedge \varsigma = 5 \wedge b\mu = 0 \wedge s_b \geq 0 \wedge 0 \leq s_c + b\mu d \leq v_{max} \rhd \delta\,{}^{c}\mathbf{0}+$	C(B7+C13)
(B8+C7)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \delta\,{}^{c}(t + d) \lhd t + d \leq u \wedge \varsigma = 3 \wedge 0 \leq \frac{1}{2}b\mu d^2 + s_b d + x_b \leq l \wedge s_c = v_{max} \rhd \delta\,{}^{c}\mathbf{0}+$	C(B8+C7)
(B8+C8)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \delta\,{}^{c}(t + d)$	C(B8+C8)

Terms resulting from encapsulating and expanding

Pair	Term	Subterm of
	$\lhd t + d \leq u \wedge \varsigma = 3 \wedge 0 \leq \frac{1}{2}b\mu d^2 + s_b d + x_b \leq l \wedge s_c \neq v_{max} \rhd \delta\text{‘}\mathbf{0}+$	
(B8+C9)	$\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \delta\text{‘}(t + d)$ $\lhd t + d \leq u + \frac{v_{max}-s_c}{b\mu} \wedge \varsigma = 4 \wedge 0 \leq \frac{1}{2}b\mu d^2 + s_b d + x_b \leq l \wedge \frac{1}{2}b\mu(\frac{v_{max}-s_c}{b\mu})^2 + s_c\frac{v_{max}-s_c}{b\mu} + x_c \leq \frac{1}{2}l \wedge b\mu \neq 0 \rhd \delta\text{‘}\mathbf{0}+$	C(B8+C9)
(B8+C10)	$\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \delta\text{‘}(t + d)$ $\lhd t + d \leq u + \frac{-s_c+\sqrt{s_c^2-2b\mu(x_c-\frac{1}{2}l)}}{b\mu} \wedge \varsigma = 4 \wedge 0 \leq \frac{1}{2}b\mu d^2 + s_b d + x_b \leq l \wedge \sqrt{s_c^2 - b\mu(2x_c - l)} \leq v_{max} \wedge b\mu \neq 0 \rhd \delta\text{‘}\mathbf{0}+$	A(B8+C10)
(B8+C11)	$\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \delta\text{‘}(t + d)$ $\lhd t + d \leq u + \frac{\frac{1}{2}l-x_c}{s_c} \wedge \varsigma = 4 \wedge 0 \leq \frac{1}{2}b\mu d^2 + s_b d + x_b \leq l \wedge b\mu = 0 \rhd \delta\text{‘}\mathbf{0}+$	A(B8+C11)

(B8+C12)	$\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \sum_{d':\mathbb{R}_{\geq \mathbf{0}}} \delta\text{‘}(t + d)$ $\lhd t + d \leq u + d' \wedge \varsigma = 5 \wedge 0 \leq \frac{1}{2}b\mu d^2 + s_b d + x_b \leq l \wedge 0 \leq s_c + b\mu d' \leq v_{max} \rhd \delta\text{‘}\mathbf{0}+$	C(B3+C12)
		C(B8+C14)

(B8+C13)	$\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \sum_{d':\mathbb{R}_{\geq \mathbf{0}}} \delta\text{‘}(t + d)$ $\lhd t + d \leq u + d' \wedge \varsigma = 5 \wedge 0 \leq \frac{1}{2}b\mu d^2 + s_b d + x_b \leq l \wedge 0 \leq s_c + b\mu d' \leq v_{max} \rhd \delta\text{‘}\mathbf{0}+$	C(B3+C12)
		C(B8+C14)

(B8+C14)	$\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \delta\text{‘}(t + d)$ $\lhd t + d \leq u + \frac{v_{max}-s_c}{b\mu} \wedge \varsigma = 5 \wedge 0 \leq \frac{1}{2}b\mu d^2 + s_b d + x_b \leq l \wedge b\mu \neq 0 \rhd \delta\text{‘}\mathbf{0}+$	C(B8+C14)

(B9+C15)	$\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \sum_{d':\mathbb{R}_{\geq \mathbf{0}}} \delta\text{‘}(t + d)$ $\lhd t + d \leq u + d' \wedge \varsigma = 6 \wedge s_b + b\mu d \geq 0 \wedge 0 \leq s_c + b\mu d' \leq v_{max} \wedge 0 \leq \frac{1}{2}b\mu d^2 + s_b d + x_b \leq \frac{1}{2}b\mu d^2 + s_b d + y_b \leq l \rhd \delta\text{‘}\mathbf{0}+$	C(B12+C15)
		C(B13+C15)

(B9+C16)	$\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \delta\text{‘}(t + d)$ $\lhd t + d \leq u + \frac{v_{max}-s_c}{b\mu} \wedge \varsigma = 6 \wedge s_b + b\mu d \geq 0 \wedge b\mu \neq 0 \wedge 0 \leq \frac{1}{2}b\mu d^2 + s_b d + x_b \leq \frac{1}{2}b\mu d^2 + s_b d + y_b \leq l \rhd \delta\text{‘}\mathbf{0}+$	C(B14+C16)
(B12+C15)	$\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \delta\text{‘}(t + \frac{-s_b+\sqrt{s_b^2-2b\mu(y_b-l)}}{b\mu})$ $\lhd t + \frac{-s_b+\sqrt{s_b^2-2b\mu(y_b-l)}}{b\mu} \leq u + d \wedge \varsigma = 6 \wedge b\mu \neq 0 \wedge \sqrt{s_b^2 - 2b\mu(y_b - l)} \geq 0 \wedge 0 \leq s_c + b\mu d \leq v_{max} \rhd \delta\text{‘}\mathbf{0}+$	C(B12+C15)

(B12+C16)	$\delta\text{‘}(t + \frac{-s_b+\sqrt{s_b^2-2b\mu(y_b-l)}}{b\mu})$ $\lhd t + \frac{-s_b+\sqrt{s_b^2-2b\mu(y_b-l)}}{b\mu} \leq u + \frac{v_{max}-s_c}{b\mu} \wedge \varsigma = 6 \wedge b\mu \neq 0 \wedge \sqrt{s_b^2 - 2b\mu(y_b - l)} \geq 0 \rhd \delta\text{‘}\mathbf{0}+$	C(B12+C15)
		C(B14+C16)

(B13+C15)	$\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \delta\text{‘}(t + \frac{l-y_b}{s_b})$ $\lhd t + \frac{l-y_b}{s_b} \leq u + d \wedge \varsigma = 6 \wedge b\mu = 0 \wedge s_b \geq 0 \wedge 0 \leq s_c + b\mu d \leq v_{max} \rhd \delta\text{‘}\mathbf{0}+$	C(B13+C15)
(B14+C15)	$\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \sum_{d':\mathbb{R}_{\geq \mathbf{0}}} \delta\text{‘}(t + d)$	C(B12+C15)

Terms resulting from encapsulating and expanding

Pair	Term	Subterm of
	$\lhd t + d \leq u + d' \wedge \varsigma = 6 \wedge 0 \leq \frac{1}{2}b\mu d^2 + s_b d + x_b \leq \frac{1}{2}b\mu d^2 + s_b d + y_b \leq l \wedge 0 \leq s_c + b\mu d' \leq v_{max} \rhd \delta\text{‹}\mathbf{0}+$	C(B13+C15)
(B14+C16)	$\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \delta\text{‹}(t+d)$ $\lhd t + d \leq u + \frac{v_{max}-s_c}{b\mu} \wedge \varsigma = 6 \wedge 0 \leq \frac{1}{2}b\mu d^2 + s_b d + x_b \leq \frac{1}{2}b\mu d^2 + s_b d + y_b \leq l \wedge b\mu \neq 0 \rhd \delta\text{‹}\mathbf{0}+$	C(B14+C16)
(C1+B1)	$\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \delta\text{‹}0$ $\lhd 0 \leq t + d \wedge \varsigma = 0 \wedge s_b + b\mu d \geq 0 \rhd \delta\text{‹}\mathbf{0}+$	A(B1+C1)
(C1+B2)	$\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \delta\text{‹}0$ $\lhd 0 \leq t + d \wedge \varsigma = 0 \rhd \delta\text{‹}\mathbf{0}+$	A(B2+C1)
(C2+B1)	$\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \sum_{d':\mathbb{R}_{\geq \mathbf{0}}} \delta\text{‹}(u+d)$ $\lhd u + d \leq t + d' \wedge \varsigma = 1 \wedge s_b + b\mu d' \geq 0$ $\wedge s_c \geq 0 \wedge 0 \leq s_c + b\mu d \leq v_{max} \rhd \delta\text{‹}\mathbf{0}+$	C(B1+C2)
(C2+B2)	$\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \sum_{d':\mathbb{R}_{\geq \mathbf{0}}} \delta\text{‹}(u+d)$ $\lhd u + d \leq t + d' \wedge \varsigma = 1$ $\wedge s_c \geq 0 \wedge 0 \leq s_c + b\mu d \leq v_{max} \rhd \delta\text{‹}\mathbf{0}+$	C(B1+C2)
(C3+B1)	$\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \delta\text{‹}(u + \frac{-s_c}{b\mu})$ $\lhd u + \frac{-s_c}{b\mu} \leq t + d \wedge \varsigma = 1 \wedge s_b + b\mu d \geq 0 \wedge b\mu \neq 0 \rhd \delta\text{‹}\mathbf{0}+$	C(B2+C3)
(C3+B2)	$\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \delta\text{‹}(u + \frac{-s_c}{b\mu})$ $\lhd u + \frac{-s_c}{b\mu} \leq t + d \wedge \varsigma = 1 \wedge b\mu \neq 0 \rhd \delta\text{‹}\mathbf{0}+$	C(B2+C3)
(C4+B1)	$\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \sum_{d':\mathbb{R}_{\geq \mathbf{0}}} \delta\text{‹}(u+d)$ $\lhd u + d \leq t + d' \wedge \varsigma = 2 \wedge s_b + b\mu d' \geq 0 \wedge 0 \leq s_c + b\mu d \leq v_{max} \wedge d \leq \Delta \rhd \delta\text{‹}\mathbf{0}+$	C(B1+C4)
(C4+B2)	$\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \sum_{d':\mathbb{R}_{\geq \mathbf{0}}} \delta\text{‹}(u+d)$ $\lhd u + d \leq t + d' \wedge \varsigma = 2 \wedge 0 \leq s_c + b\mu d \leq v_{max} \wedge d \leq \Delta \rhd \delta\text{‹}\mathbf{0}+$	C(B1+C4)
(C5+B1)	$\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \delta\text{‹}(u + \frac{v_{max}-s_c}{b\mu})$ $\lhd u + \frac{v_{max}-s_c}{b\mu} \leq t + d \wedge \varsigma = 2 \wedge s_b + b\mu d \geq 0 \wedge b\mu \neq 0 \wedge \frac{v_{max}-s_c}{b\mu} \leq \Delta \rhd \delta\text{‹}\mathbf{0}+$	C(B2+C5)
(C5+B2)	$\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \delta\text{‹}(u + \frac{v_{max}-s_c}{b\mu})$ $\lhd u + \frac{v_{max}-s_c}{b\mu} \leq t + d \wedge \varsigma = 2 \wedge b\mu \neq 0 \wedge \frac{v_{max}-s_c}{b\mu} \leq \Delta \rhd \delta\text{‹}\mathbf{0}+$	C(B2+C5)
(C6+B1)	$\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \delta\text{‹}(u + \Delta)$ $\lhd u + \Delta \leq t + d \wedge \varsigma = 2 \wedge s_b + b\mu d \geq 0 \wedge 0 \leq s_c + b\mu\Delta \leq v_{max} \rhd \delta\text{‹}\mathbf{0}+$	C(B2+C6)
(C6+B2)	$\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \delta\text{‹}(u + \Delta)$ $\lhd u + \Delta \leq t + d \wedge \varsigma = 2 \wedge 0 \leq s_c + b\mu\Delta \leq v_{max} \rhd \delta\text{‹}\mathbf{0}+$	C(B2+C6)
(C7+B3)	$\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \delta\text{‹}u$ $\lhd u \leq t + d \wedge \varsigma = 3 \wedge s_b + b\mu d \geq 0 \wedge s_c = v_{max} \wedge 0 \leq \frac{1}{2}b\mu d^2 + s_b d + x_b \leq l \rhd \delta\text{‹}\mathbf{0}+$	C(B8+C7)
(C7+B4)	$\delta\text{‹}u$	C(B8+C7)

Terms resulting from encapsulating and expanding

Pair	Term	Subterm of
	$\lhd u \leq t+\frac{-s_b-\sqrt{s_b^2-2b\mu x_b}}{b\mu} \wedge \varsigma = 3 \wedge b\mu \neq 0 \wedge \sqrt{s_b^2-2b\mu x_b} > 0 \wedge s_c = v_{\mathit{max}} \rhd \delta\text{ˁ}\mathbf{0}+$	
(C7+B5)	$\delta\text{ˁ}u$ $\lhd u \leq t+\frac{-x_b}{s_b} \wedge \varsigma = 3 \wedge b\mu = 0 \wedge s_b < 0 \wedge s_c = v_{\mathit{max}} \rhd \delta\text{ˁ}\mathbf{0}+$	C(B8+C7)
(C7+B6)	$\delta\text{ˁ}u$ $\lhd u \leq t+\frac{-s_b+\sqrt{s_b^2-2b\mu(x_b-l)}}{b\mu} \wedge \varsigma = 3 \wedge b\mu \neq 0 \wedge \sqrt{s_b^2-2b\mu(x_b-l)} \geq 0 \wedge s_c = v_{\mathit{max}} \rhd \delta\text{ˁ}\mathbf{0}+$	C(B8+C7)
(C7+B7)	$\delta\text{ˁ}u$ $\lhd u \leq t+\frac{l-x_b}{s_b} \wedge \varsigma = 3 \wedge b\mu = 0 \wedge s_b \geq 0 \wedge s_c = v_{\mathit{max}} \rhd \delta\text{ˁ}\mathbf{0}+$	C(B8+C7)
(C7+B8)	$\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \delta\text{ˁ}u$ $\lhd u \leq t+d \wedge \varsigma = 3 \wedge 0 \leq \frac{1}{2}b\mu d^2 + s_b d + x_b \leq l \wedge s_c = v_{\mathit{max}} \rhd \delta\text{ˁ}\mathbf{0}+$	C(B8+C7)
(C8+B3)	$\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \delta\text{ˁ}u$ $\lhd u \leq t+d \wedge \varsigma = 3 \wedge s_b + b\mu d \geq 0 \wedge s_c \neq v_{\mathit{max}} \wedge 0 \leq \frac{1}{2}b\mu d^2 + s_b d + x_b \leq l \rhd \delta\text{ˁ}\mathbf{0}+$	C(B8+C8)
(C8+B4)	$\delta\text{ˁ}u$ $\lhd u \leq t+\frac{-s_b-\sqrt{s_b^2-2b\mu x_b}}{b\mu} \wedge \varsigma = 3 \wedge b\mu \neq 0 \wedge \sqrt{s_b^2-2b\mu x_b} > 0 \wedge s_c \neq v_{\mathit{max}} \rhd \delta\text{ˁ}\mathbf{0}+$	C(B8+C8)
(C8+B6)	$\delta\text{ˁ}u$ $\lhd u \leq t+\frac{-s_b+\sqrt{s_b^2-2b\mu(x_b-l)}}{b\mu} \wedge \varsigma = 3 \wedge b\mu \neq 0 \wedge \sqrt{s_b^2-2b\mu(x_b-l)} \geq 0 \wedge s_c \neq v_{\mathit{max}} \rhd \delta\text{ˁ}\mathbf{0}+$	C(B8+C8)
(C8+B7)	$\delta\text{ˁ}u$ $\lhd u \leq t+\frac{l-x_b}{s_b} \wedge \varsigma = 3 \wedge b\mu = 0 \wedge s_b \geq 0 \wedge s_c \neq v_{\mathit{max}} \rhd \delta\text{ˁ}\mathbf{0}+$	C(B8+C8)
(C8+B8)	$\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \delta\text{ˁ}u$ $\lhd u \leq t+d \wedge \varsigma = 3 \wedge 0 \leq \frac{1}{2}b\mu d^2 + s_b d + x_b \leq l \wedge s_c \neq v_{\mathit{max}} \rhd \delta\text{ˁ}\mathbf{0}+$	C(B8+C8)
(C9+B3)	$\sum_{d:\mathbb{R}_{\geq \mathbf{0}}} \delta\text{ˁ}(u+\frac{v_{\mathit{max}}-s_c}{b\mu})$ $\lhd u+\frac{v_{\mathit{max}}-s_c}{b\mu} \leq t+d \wedge \varsigma = 4 \wedge s_b + b\mu d \geq 0 \wedge \frac{1}{2}b\mu(\frac{v_{\mathit{max}}-s_c}{b\mu})^2 + s_c\frac{v_{\mathit{max}}-s_c}{b\mu} + x_c \leq \frac{1}{2}l \wedge b\mu \neq 0 \wedge 0 \leq \frac{1}{2}b\mu d^2 + s_b d + x_b \leq l \rhd \delta\text{ˁ}\mathbf{0}+$	C(B8+C9)
(C9+B4)	$\delta\text{ˁ}(u+\frac{v_{\mathit{max}}-s_c}{b\mu})$ $\lhd u+\frac{v_{\mathit{max}}-s_c}{b\mu} \leq t+\frac{-s_b-\sqrt{s_b^2-2b\mu x_b}}{b\mu} \wedge \varsigma = 4 \wedge b\mu \neq 0 \wedge \sqrt{s_b^2-2b\mu x_b} > 0 \wedge \frac{1}{2}b\mu(\frac{v_{\mathit{max}}-s_c}{b\mu})^2 + s_c\frac{v_{\mathit{max}}-s_c}{b\mu} + x_c \leq \frac{1}{2}l \rhd \delta\text{ˁ}\mathbf{0}+$	C(B8+C9)
(C9+B6)	$\delta\text{ˁ}(u+\frac{v_{\mathit{max}}-s_c}{b\mu})$	C(B8+C9)

Terms resulting from encapsulating and expanding

Pair	Term	Subterm of
	$\lhd u + \frac{v_{max}-s_c}{b\mu} \leq t + \frac{-s_b+\sqrt{s_b^2-2b\mu(x_b-l)}}{b\mu} \wedge \varsigma = 4 \wedge b\mu \neq 0 \wedge \sqrt{s_b^2 - 2b\mu(x_b - l)} \geq 0 \wedge \frac{1}{2}b\mu(\frac{v_{max}-s_c}{b\mu})^2 + s_c\frac{v_{max}-s_c}{b\mu} + x_c \leq \frac{1}{2}l \rhd \delta^{\text{c}}\mathbf{0}+$	
(C9+B8)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \delta^{\text{c}}(u + \frac{v_{max}-s_c}{b\mu}) \lhd u + \frac{v_{max}-s_c}{b\mu} \leq t + d \wedge \varsigma = 4 \wedge 0 \leq \frac{1}{2}b\mu d^2 + s_b d + x_b \leq l \wedge \frac{1}{2}b\mu(\frac{v_{max}-s_c}{b\mu})^2 + s_c\frac{v_{max}-s_c}{b\mu} + x_c \leq \frac{1}{2}l \wedge b\mu \neq 0 \rhd \delta^{\text{c}}\mathbf{0}+$	C(B8+C9)
(C12+B3)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \sum_{d':\mathbb{R}_{\geq\mathbf{0}}} \delta^{\text{c}}(u + d) \lhd u + d \leq t + d' \wedge \varsigma = 5 \wedge s_b + b\mu d' \geq 0 \wedge 0 \leq s_c + b\mu d \leq v_{max} \wedge 0 \leq \frac{1}{2}b\mu d^2 + s_b d + x_b \leq l \rhd \delta^{\text{c}}\mathbf{0}+$	C(B3+C12)
(C12+B4)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \delta^{\text{c}}(u + d) \lhd u + d \leq t + \frac{-s_b-\sqrt{s_b^2-2b\mu x_b}}{b\mu} \wedge \varsigma = 5 \wedge b\mu \neq 0 \wedge \sqrt{s_b^2 - 2b\mu x_b} > 0 \wedge 0 \leq s_c + b\mu d \leq v_{max} \rhd \delta^{\text{c}}\mathbf{0}+$	f
(C12+B6)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \delta^{\text{c}}(u + d) \lhd u + d \leq t + \frac{-s_b+\sqrt{s_b^2-2b\mu(x_b-l)}}{b\mu} \wedge \varsigma = 5 \wedge b\mu \neq 0 \wedge \sqrt{s_b^2 - 2b\mu(x_b - l)} \geq 0 \wedge 0 \leq s_c + b\mu d \leq v_{max} \rhd \delta^{\text{c}}\mathbf{0}+$	C(B3+C12) C(B8+C14)
(C12+B7)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \delta^{\text{c}}(u + d) \lhd u + d \leq t + \frac{l-x_b}{s_b} \wedge \varsigma = 5 \wedge b\mu = 0 \wedge s_b \geq 0 \wedge 0 \leq s_c + b\mu d \leq v_{max} \rhd \delta^{\text{c}}\mathbf{0}+$	C(B7+C13)
(C12+B8)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \sum_{d':\mathbb{R}_{\geq\mathbf{0}}} \delta^{\text{c}}(u + d) \lhd u + d \leq t + d' \wedge \varsigma = 5 \wedge 0 \leq \frac{1}{2}b\mu d'^2 + s_b d' + x_b \leq l \wedge 0 \leq s_c + b\mu d \leq v_{max} \rhd \delta^{\text{c}}\mathbf{0}+$	C(B3+C12)
(C13+B3)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \sum_{d':\mathbb{R}_{\geq\mathbf{0}}} \delta^{\text{c}}(u + d) \lhd u + d \leq t + d' \wedge \varsigma = 5 \wedge s_b + b\mu d' \geq 0 \wedge 0 \leq s_c + b\mu d \leq v_{max} \wedge 0 \leq \frac{1}{2}b\mu d'^2 + s_b d' + x_b \leq l \rhd \delta^{\text{c}}\mathbf{0}+$	C(B3+C13)
(C13+B4)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \delta^{\text{c}}(u + d) \lhd u + d \leq t + \frac{-s_b-\sqrt{s_b^2-2b\mu x_b}}{b\mu} \wedge \varsigma = 5 \wedge b\mu \neq 0 \wedge \sqrt{s_b^2 - 2b\mu x_b} > 0 \wedge 0 \leq s_c + b\mu d \leq v_{max} \rhd \delta^{\text{c}}\mathbf{0}+$	f
(C13+B6)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \delta^{\text{c}}(u + d) \lhd u + d \leq t + \frac{-s_b+\sqrt{s_b^2-2b\mu(x_b-l)}}{b\mu} \wedge \varsigma = 5 \wedge b\mu \neq 0 \wedge \sqrt{s_b^2 - 2b\mu(x_b - l)} \geq 0 \wedge 0 \leq s_c + b\mu d \leq v_{max} \rhd \delta^{\text{c}}\mathbf{0}+$	C(B6+C13)
(C13+B7)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \delta^{\text{c}}(u + d) \lhd u + d \leq t + \frac{l-x_b}{s_b} \wedge \varsigma = 5 \wedge b\mu = 0 \wedge s_b \geq 0 \wedge 0 \leq s_c + b\mu d \leq v_{max} \rhd \delta^{\text{c}}\mathbf{0}+$	C(B7+C13)
(C13+B8)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \sum_{d':\mathbb{R}_{\geq\mathbf{0}}} \delta^{\text{c}}(u + d) \lhd u + d \leq t + d' \wedge \varsigma = 5 \wedge 0 \leq \frac{1}{2}b\mu d'^2 + s_b d' + x_b \leq l \wedge 0 \leq s_c + b\mu d \leq v_{max} \rhd \delta^{\text{c}}\mathbf{0}+$	C(B3+C12)

Terms resulting from encapsulating and expanding

Pair	Term	Subterm of
(C14+B3)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \delta{\scriptscriptstyle c}(u+\frac{v_{\mathit{max}}-s_c}{b\mu}) \lhd u+\frac{v_{\mathit{max}}-s_c}{b\mu} \leq t+d \wedge \varsigma = 5 \wedge s_b + b\mu d \geq 0 \wedge b\mu \neq 0 \wedge 0 \leq \frac{1}{2}b\mu d^2 + s_b d + x_b \leq l \rhd \delta{\scriptscriptstyle c}\mathbf{0}+$	C(B8+C14)
(C14+B4)	$\delta{\scriptscriptstyle c}(u+\frac{v_{\mathit{max}}-s_c}{b\mu}) \lhd u+\frac{v_{\mathit{max}}-s_c}{b\mu} \leq t+\frac{-s_b-\sqrt{s_b^2-2b\mu x_b}}{b\mu} \wedge \varsigma = 5 \wedge b\mu \neq 0 \wedge \sqrt{s_b^2-2b\mu x_b} > 0 \rhd \delta{\scriptscriptstyle c}\mathbf{0}+$	f

(C14+B6)	$\delta{\scriptscriptstyle c}(u+\frac{v_{\mathit{max}}-s_c}{b\mu}) \lhd u+\frac{v_{\mathit{max}}-s_c}{b\mu} \leq t+\frac{-s_b+\sqrt{s_b^2-2b\mu(x_b-l)}}{b\mu} \wedge \varsigma = 5 \wedge b\mu \neq 0 \wedge \sqrt{s_b^2-2b\mu(x_b-l)} \geq 0 \rhd \delta{\scriptscriptstyle c}\mathbf{0}+$	C(B8+C4)
		C(B6+C13)

(C14+B8)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \delta{\scriptscriptstyle c}(u+\frac{v_{\mathit{max}}-s_c}{b\mu}) \lhd u+\frac{v_{\mathit{max}}-s_c}{b\mu} \leq t+d \wedge \varsigma = 5 \wedge 0 \leq \frac{1}{2}b\mu d^2 + s_b d + x_b \leq l \wedge b\mu \neq 0 \rhd \delta{\scriptscriptstyle c}\mathbf{0}+$	C(B8+C14)

(C15+B9)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \sum_{d':\mathbb{R}_{\geq\mathbf{0}}} \delta{\scriptscriptstyle c}(u+d) \lhd u+d \leq t+d' \wedge \varsigma = 6 \wedge s_b + b\mu d' \geq 0 \wedge 0 \leq s_c + b\mu d \leq v_{\mathit{max}} \wedge 0 \leq \frac{1}{2}b\mu d'^2 + s_b d' + x_b \leq \frac{1}{2}b\mu d'^2 + s_b d' + y_b \leq l \rhd \delta{\scriptscriptstyle c}\mathbf{0}+$	C(B12+C15)
		C(B13+C15)

(C15+B12)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \delta{\scriptscriptstyle c}(u+d) \lhd u+d \leq t+\frac{-s_b+\sqrt{s_b^2-2b\mu(y_b-l)}}{b\mu} \wedge \varsigma = 6 \wedge b\mu \neq 0 \wedge \sqrt{s_b^2-2b\mu(y_b-l)} \geq 0 \wedge 0 \leq s_c + b\mu d \leq v_{\mathit{max}} \rhd \delta{\scriptscriptstyle c}\mathbf{0}+$	C(B12+C15)
(C15+B13)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \delta{\scriptscriptstyle c}(u+d) \lhd u+d \leq t+\frac{l-y_b}{s_b} \wedge \varsigma = 6 \wedge b\mu = 0 \wedge s_b \geq 0 \wedge 0 \leq s_c + b\mu d \leq v_{\mathit{max}} \rhd \delta{\scriptscriptstyle c}\mathbf{0}+$	C(B13+C15)

(C15+B14)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \sum_{d':\mathbb{R}_{\geq\mathbf{0}}} \delta{\scriptscriptstyle c}(u+d) \lhd u+d \leq t+d' \wedge \varsigma = 6 \wedge 0 \leq \frac{1}{2}b\mu d'^2 + s_b d' + x_b \leq \frac{1}{2}b\mu d'^2 + s_b d' + y_b \leq l \wedge 0 \leq s_c + b\mu d \leq v_{\mathit{max}} \rhd \delta{\scriptscriptstyle c}\mathbf{0}+$	C(B12+C15)
		C(B13+C15)

(C16+B9)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \delta{\scriptscriptstyle c}(u+\frac{v_{\mathit{max}}-s_c}{b\mu}) \lhd u+\frac{v_{\mathit{max}}-s_c}{b\mu} \leq t+d \wedge \varsigma = 6 \wedge s_b + b\mu d \geq 0 \wedge b\mu \neq 0 \wedge 0 \leq \frac{1}{2}b\mu d^2 + s_b d + x_b \leq \frac{1}{2}b\mu d^2 + s_b d + y_b \leq l \rhd \delta{\scriptscriptstyle c}\mathbf{0}+$	C(B14+C16)
		C(B12+C15)

(C16+B12)	$\delta{\scriptscriptstyle c}(u+\frac{v_{\mathit{max}}-s_c}{b\mu}) \lhd u+\frac{v_{\mathit{max}}-s_c}{b\mu} \leq t+\frac{-s_b+\sqrt{s_b^2-2b\mu(y_b-l)}}{b\mu} \wedge \varsigma = 6 \wedge b\mu \neq 0 \wedge \sqrt{s_b^2-2b\mu(y_b-l)} \geq 0 \rhd \delta{\scriptscriptstyle c}\mathbf{0}+$	C(B12+C15)

(C16+B14)	$\sum_{d:\mathbb{R}_{\geq\mathbf{0}}} \delta{\scriptscriptstyle c}(u+\frac{v_{\mathit{max}}-s_c}{b\mu}) \lhd u+\frac{v_{\mathit{max}}-s_c}{b\mu} \leq t+d \wedge \varsigma = 6 \wedge 0 \leq \frac{1}{2}b\mu d^2 + s_b d + x_b \leq \frac{1}{2}b\mu d^2 + s_b d + y_b \leq l \wedge b\mu \neq 0 \rhd \delta{\scriptscriptstyle c}\mathbf{0}+$	C(B12+C15)
		C(B14+C16)

Terms resulting from encapsulating and expanding

Bibliography

[1] P. Abdulla, A. Bouajjani, and B. Jonsson. On-the-fly analysis of systems with unbounded, lossy FIFO channels. In A.J. Hu and M.Y. Vardi, editors, *10th International Conference on Computer Aided Verification, CAV'98*, volume 1427 of *Lecture Notes in Computer Science*, pages 305–318. Springer-Verlag, 1998.

[2] K. Aben, P. van den Brand, and R. Schouten. Specification of the EUV wafer handler controller, July 2002. Eindhoven University of Technology.

[3] L. de Alfaro, T.A. Henzinger, and F.Y.C. Mang. The control of synchronous systems. In C. Palamidessi, editor, *Proceedings of the 11th International Conference on Concurrency Theory (CONCUR 00)*, volume 1877 of *Lecture Notes in Computer Science*, pages 458–473. Springer-Verlag, 2000.

[4] L. de Alfaro, T.A. Henzinger, and F.Y.C. Mang. The control of synchronous systems, part II. In K.G. Larsen and M. Nielsen, editors, *Proceedings of the 12th International Conference on Concurrency Theory (CONCUR 01)*, volume 2154 of *Lecture Notes in Computer Science*, pages 566–580. Springer-Verlag, 2001.

[5] R. Alur. Timed automata. In N. Halbwachs and D. Peled, editors, *Computer Aided Verification, 11th International Conference, CAV'99*, volume 1633 of *Lecture Notes in Computer Science*, pages 8–22. Springer-Verlag, 1999. Also appeared in NATO-ASI 1998 Summer School on Verification of Digital and Hybrid Systems.

[6] R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.

[7] R. Alur and D.L. Dill. Automata for modeling real-time systems. In M. Paterson, editor, *Automata, Languages and Programming: Proceedings of the 17th ICALP*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. Springer-Verlag, 1990.

[8] R. Alur and D.L. Dill. A theory of timed automata. In *Theoretical Computer Science*, volume 126, pages 183–235, 1994.

[9] R. Alur, T.A. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. *IEEE Transactions on Software Engineering*, 22(3):181–201, 1996.

[10] S. Andova. *Probabilistic Process Algebra*. PhD thesis, Eindhoven University of Technology, November 2002.

[11] J.C.M. Baeten. Embedding untimed into timed process algebra; the case for explicit termination. *Mathematical Structures in Computer Science*, 2003. To appear.

[12] J.C.M. Baeten and J.A. Bergstra. Real time process algebra. *Formal Aspects of Computing*, 3(2):142–188, 1991.

[13] J.C.M. Baeten and J.A. Bergstra. Real space process algebra. *Formal Aspects of Computing*, 5(6):481–529, 1993.

[14] J.C.M. Baeten and J.A. Bergstra. Discrete time process algebra. *Formal Aspects of Computing*, 8(2):188–208, 1996.

[15] J.C.M. Baeten and J.A. Bergstra. Discrete time process algebra: absolute time, relative time and parametric time. *Fundamenta Informaticae*, 29(1-2):51–76, 1997.

[16] J.C.M. Baeten, J.A. Bergstra, and M.A. Reniers. Discrete time process algebra with silent step. In G. Plotkin, C. Stirling, and M. Tofte, editors, *Proof, Language, and Interaction: Essays in Honour of Robin Milner*, Fountations of Computing Series, chapter 18, pages 535–569. MIT Press, 2000.

[17] J.C.M. Baeten and C.A. Middelburg. *Process Algebra with Timing*. EATCS Monograph. Springer-Verlag, 2002.

[18] J.C.M. Baeten and W.P. Weijland. *Process Algebra*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1990.

[19] B. Bérard and C. Dufourd. Timed automata and additive clock constraints. *Information Processing Letters*, 75(1-2):1–7, 2000.

[20] M.A. Bezem and J.F. Groote. Invariants in process algebra with data. In B. Jonsson and J. Parrow, editors, *Proceedings Concur'94, Uppsala, Sweden*, volume 836 of *Lecture Notes in Computer Science*, pages 401–416. Springer-Verlag, 1994.

[21] S.C.C. Blom, W.J. Fokkink, J.F. Groote, I. Van Langevelde, B.Lisser, and J.C. van de Pol. μCRL: A toolset for analysing algebraic specification. In *13th International Conference on Computer Aided Verification, CAV'01*, volume 2102 of *Lecture Notes in Computer Science*, pages 250–254. Springer-Verlag, 2001.

[22] J.-P. Bodeveix and M. Filali. FMona: A tool for expressing validation techniques over infinite state systems. In S. Graf and M. Schwartzbach, editors, *Proceedings of the 6th conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2000)*, volume 1785 of *Lecture Notes in Computer Science*, pages 204–219. Springer-Verlag, 2000.

[23] B. Boigelot, P. Godefroid, B. Willems, and P. Wolper. The power of QDDs. In P. van Hentenryck, editor, *Static Analysis, 4th International Symposium, SAS '97*, volume 1302 of *Lecture Notes in Computer Science*, pages 172–186. Springer-Verlag, 1997.

[24] T. Bolognesi and E. Brinksma. Introduction to the ISO specification language LOTOS. In *Computer Networks and ISDN Systems*, volume 14, pages 25–59, 1987.

[25] S. Bornot, S. Sifakis, and S. Tripakis. Modeling urgency in timed systems. In W.P. de Roever, H. Langmaack, and A. Pnueli, editors, *Compositionality: The Significant Difference, International Symposium, COMPOS'97*, volume 1536 of *Lecture Notes in Computer Science*, pages 103–129. Springer-Verlag, 1998.

[26] V. Bos and J.J.T. Kleijn. *Formal Specification and Analysis of Industrial Systems*. PhD thesis, Eindhoven University of Technology, March 2002.

[27] D. Bošnački. *Enhancing State Space Reduction Techniques for Model Checking*. PhD thesis, Eindhoven University of Technology, November 2001.

[28] P. Bouyer, C. Dufourd, E. Fleury, and A. Petit. Are timed automata updatable? In *12th International Conference on Computer Aided Verification, CAV'00*, volume 1855 of *Lecture Notes in Computer Science*, pages 464–479. Springer-Verlag, 2000.

[29] J.C Bradfield and C. Stirling. Modal logics and mu-calculi. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, chapter 4, pages 293–330. Elsevier (North-Holland), 2001.

[30] R.E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[31] R.E. Bryant, S.K. Lahiri, and S.A. Seshia. Modeling and verifying systems using a logic of counter arithmetic with lambda expressions and uninterpreted functions. In *14th International Conference on Computer Aided Verification, CAV 2002*, volume 2404 of *Lecture Notes in Computer Science*, pages 78–92. Springer-Verlag, 2002.

[32] T. Bultan, R. Gerber, and W. Pugh. Symbolic model checking of infinite state systems using presburger arithmetic. In O. Grumberg, editor, *Proceedings of the 9^{th} International Conference on Computer Aided Verification (CAV'97)*, volume 1254 of *Lecture Notes in Computer Science*, pages 400–411. Springer-Verlag, June 1997.

[33] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.

[34] F. Cassez and K.G. Larsen. The impressive power of stopwatches. In C. Palamidessi, editor, *CONCUR 2000 - Concurrency Theory*, volume 1877 of *Lecture Notes in Computer Science*, pages 138–152. Springer-Verlag, 2000.

[35] K. Čerāns. *Algorithmic Problems in Analysis of Real-Time System Specifications*. PhD thesis, University of Latvia, 1992.

[36] L. Chen. *Timed Processes: Models, Axioms and Decidability*. PhD thesis, University of Edinburgh, 1992.

[37] P.J.L. Cuijpers and M.A. Reniers. Topological (bi)-simulation. Technical Report CSR 02-04, Eindhoven University of Technology, Department of Mathemantics and Computer Science, 2002.

[38] P.R. D'Argenio. *Algebras and Automata for Timed and Stochastic Systems*. PhD thesis, Twente University, November 1999.

[39] N. Dershowitz and D.A. Plaisted. Rewriting. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 9, pages 535–610. Elsevier Science, 2001.

[40] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics*. EATCS-Monographs on Theoretical Computer Science. Springer-Verlag, 1985.

[41] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 2: Module Specifications and Constraints*. EATCS-Monographs on Theoretical Computer Science. Springer-Verlag, 1990.

[42] E.A. Emerson and K.S. Namjoshi. Automatic verification of parameterized synchronous systems. In R. Alur and T. Henzinger, editors, *Proceedings of the 8th International Conference on Computer Aided Verification (CAV'96)*, volume 1102 of *Lecture Notes in Computer Science*, pages 87–98. Springer-Verlag, 1996.

[43] W.J. Fokkink. An elimination theorem for regular behaviours with integration. In E. Best, editor, *Proc. 4th Conference on Concurrency Theory (CONCUR'93)*, pages 432–446. Springer-Verlag, 1993.

[44] W.J. Fokkink. *Clocks, Trees and Stars in Process Theory*. PhD thesis, University of Amsterdam, December 1994.

[45] H. Garavel. An overview of the EUCALYPTUS toolbox. In Z. Brezočnik and T. Kapus, editors, *Proceedings of the COST 247 International Workshop on Applied Formal Methods in System Design (Maribor, Slovenia)*, pages 76–88. University of Maribor, 1996.

[46] H. Garavel. OPEN/CAESAR: An Open Software Architecture for Verification, Simulation, and Testing. In B. Steffen, editor, *Proceedings of the Fourth International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS'98*, volume 1384 of *Lecture Notes in Computer Science*, pages 68–84. Springer-Verlag, March 1998.

[47] H. Garavel and M. Sighireanu. Towards a second generation of formal description techniques — rationale for the design of E-LOTOS. In J.F. Groote, S.P. Luttik, and J.J. van Wamel, editors, *Proceedings of FMICS'98, Amsterdam*. CWI, 1998.

[48] R.J. van Glabbeek. The linear time – branching time spectrum I. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, chapter 1, pages 5–99. Elsevier (North-Holland), 2001.

[49] J.F. Groote. The syntax and semantics of timed μCRL. Software Engineering Report SEN-R9709, CWI, June 1997.

[50] J.F. Groote and S.P. Luttik. Undecidability and completeness results for process algebras with alternative quantification over data. Software Engineering Report SEN-R9806, CWI, June 1998.

[51] J.F. Groote and R. Mateescu. Verification of temporal properties of processes in a setting with data. In A.M. Haeberer, editor, *Proceedings of the 7th International Conference on Algebraic Methodology and Software Technology AMAST'98 (Amazonia, Brazil)*, volume 1548 of *Lecture Notes in Computer Science*, pages 74–90. Springer-Verlag, January 1999.

[52] J.F. Groote and A. Ponse. The syntax and semantics of μCRL. In A. Ponse, C. Verhoef, and S.F.M. van Vlijmen, editors, *Algebra of Communicating Processes '94*, Workshops in Computing Series, pages 26–62. Springer Verlag, 1995.

[53] J.F. Groote and M.A. Reniers. Algebraic process verification. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, chapter 17, pages 1151–1208. Elsevier (North-Holland), 2001.

[54] J.F. Groote and J. Springintveld. Focus points and convergent operators: a proof strategy for protocol verification. *The Journal of Logic and Algebraic Programming*, 49:31–60, 2001.

[55] J.F. Groote and J.C. van de Pol. A bounded retransmission protocol for large data packets. a case study in computer checked verification. In M. Wirsing and M. Nivat, editors, *Proceedings of AMAST'96, Munich*, volume 110 of *Lecture Notes in Computer Science*, pages 536–550. Springer-Verlag, 1996.

[56] J.F. Groote and J.C. van der Pol. Equational binary decision diagrams. In M. Parigot and A. Voronkov, editors, *Logic for Programming and Reasoning, LPAR2000*, volume 1955 of *Lecture Notes in Artificial Intelligence*, pages 161–178. Springer-Verlag, 2000.

[57] J.F. Groote and J.J. van Wamel. Basic theorems for parallel processes in timed μCRL. Software Engineering Report SEN-R9808, CWI, June 1998.

[58] J.F. Groote and J.J. van Wamel. Analysis of three hybrid systems in timed μCRL. *Journal of Logic and Algebraic Programming*, 39:215–247, 2001.

[59] J.F. Groote and T.A.C. Willemse. A checker for modal formulas for processes with data. Technical Report CSR 02-16, Eindhoven University of Technology, Department of Mathematics and Computer Science, 2002. Submitted for publication.

[60] Y. Gurevich and L. Harrington. Trees, automata, and games. In *Proceedings of the 14th annual ACM symposium on Theory of Computing*, pages 60–65. ACM Press, 1982.

[61] P.R. Halmos. *Measure Theory*. Springer, 1974.

[62] T.A. Henzinger. The theory of hybrid automata. In *the Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS 96)*, pages 278–292, 1996.

[63] T.A. Henzinger, B. Horowitz, and R. Majumdar. Rectangular hybrid games. In *CONCUR: Concurrency Theory*, volume 1664 of *Lecture Notes in Computer Science*, pages 320–335. Springer-Verlag, 1999.

[64] T.A. Henzinger, P.W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57:94–124, 1998.

[65] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111:193–244, 1994.

[66] T.A. Henzinger and J.-F. Raskin. Robust undecidability of timed and hybrid systems. In N. Lynch and B.H. Krogh, editors, *Proceedings of the Third International Workshop on Hybrid Systems: Computation and Control (HSCC'00)*, volume 1790 of *Lecture Notes in Computer Science*, pages 145–159, 2000.

[67] G. Holzmann and D. Peled. An improvement in formal verification. In D. Hogrefe and S. Leue, editors, *FORTE 1994, Bern, Switzerland*, volume 6 of *IFIP Conference Proceedings*, pages 197–211. Chapman & Hall, 1994.

[68] ISO/IEC. *Information technology – Enhancements to* LOTOS *(*E-LOTOS*)*, volume ISO/IEC IS 15437:2001. ISO/IEC, July 2001.

[69] ITU-T. *Recommendation Z.100 Annex I: SDL Methodology Guidelines*. ITU-T, Geneva, 1993.

[70] ITU-T. *Recommendation Z.120: Message Sequence Chart (MSC)*. ITU-T, Geneva, September 1993.

[71] ITU-T. *Recommendation Z.100: Specification and Description Language (SDL)*. ITU-T, Geneva, June 1994.

[72] A.S. Jeffrey. A linear time process algebra. In K.G. Larsen and A. Skou, editors, *Proceedings CAV'91*, volume 575 of *Lecture Notes in Computer Science*, pages 433–442. Springer-Verlag, 1992.

[73] A.S. Klusener. *Models and Axioms for a Fragment of Real Time Process Algebra*. PhD thesis, Eindhoven University of Technology, December 1993.

[74] H.P. Korver and J. Springintveld. A computer-checked verification of Milner's scheduler. In M. Hagiya and J.C. Mitchell, editors, *Proceedings of the International Symposium on Theoretical Aspects of Computer Software (TACS'94)*, volume 789 of *Lecture Notes in Computer Science*, pages 161–178. Springer-Verlag, 1994.

[75] D. Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[76] O. Kupferman, P. Madhusudan, P.S. Thiagarajan, and M.Y. Vardi. Open systems in reactive environments: Control and synthesis. In C. Palamidessi, editor, *Proceedings of the* 11^{th} *International Conference on Concurrency Theory*, volume 1877 of *Lecture Notes in Computer Science*, pages 92–107, 2000.

[77] O. Kupferman and M.Y. Vardi. Synthesis with incomplete information. In 2^{nd} *International Conference on Temporal Logic*, pages 91–106, July 1997.

[78] O. Kupferman and M.Y. Vardi. μ-calculus synthesis. In M. Nielsen and B. Rovan, editors, *Proc. 25th International Symposium on Mathematical Foundations of Computer Science*, volume 1893 of *Lecture Notes in Computer Science*, pages 497–507. Springer-Verlag, 2000.

[79] K. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *Springer International Journal of Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.

[80] J.-L. Lassez, V.L. Nguyen, and E.A. Sonenberg. Fixed point theorems and semantics: a folk tale. *Information Processing Letters*, 14(3):112–116, May 1988.

[81] G. Leduc. An upward compatible timed extension to LOTOS. In K. Parther and G. Rose, editors, *Proceedings of FORTE'91, November, Sydney, Australia*, Formal Description Techniques IV, IFIP Transactions C-2, pages 217–232. Elsevier Science, 1991.

[82] L. Léonard and G. Leduc. A formal definition of time in LOTOS – extended abstract. *Formal Aspects of Computing*, 10(3):248–266, 1998.

[83] H. Lin and W. Yi. Axiomatising timed automata. *Acta Informatica*, 38:277–305, 2002.

[84] S.P. Luttik. *Choice Quantification in Process Algebra*. PhD thesis, University of Amsterdam, April 2002.

[85] N. Lynch, R. Segala, F. Vaandrager, and H.B. Weinberg. Hybrid I/O automata. In R. Alur, T.A. Henzinger, and E.D. Sontag, editors, *Hybrid Systems III*, volume 1066 of *Lecture Notes in Computer Science*, pages 496–510. Springer-Verlag, 1996.

[86] N. Lynch, R. Segala, and F.W. Vaandrager. Hybrid I/O automata revisited. In M.D. Di Benedetto and A.L. Sangiovanni-Vincentelli, editors, *Proceedings of the Fourth International Workshop on Hybrid Systems: Computation and Control (HSCC'01)*, volume 2034 of *Lecture Notes in Computer Science*, pages 403–417. Springer-Verlag, 2001.

[87] N. Lynch, R. Segala, and F.W. Vaandrager. Hybrid I/O automata. Nijmegen Institute for Computing and Information Sciences Technical Report NIII-R0201, Katholieke Universiteit Nijmegen, 2002.

[88] A. Mader. *Verification of Modal Properties Using Boolean Equation Systems*. PhD thesis, Technical University of Munich, 1997.

[89] P. Madhusudan and P.S. Thiagarajan. A decidable class of asynchronous distributed controllers. In L. Brim, P. Jancar, M. Kretínský, and A. Kucera, editors, *CONCUR 2002 - Concurrency Theory*, volume 2421 of *Lecture Notes in Computer Science*, pages 145–160. Springer-Verlag, 2002.

[90] L.E. Mamane, L. Cleophas, and E. Smeets. Specification & analysis of embedded systems assignment report, July 2002. Eindhoven University of Technology.

[91] R. Mateescu. Local model-checking of an alternation-free value-based modal mu-calculus. In A. Bossi, A. Cortesi, and F. Levi, editors, *Proceedings of the 2^{nd} International Workshop on Verification, Model Checking and Abstract Interpretation*. University Ca' Foscari of Venice, September 1998.

[92] R. Mateescu. *Vérification des propriétés temporelles des programmes parallèles*. PhD thesis, Institut National Polytechnique de Grenoble, April 1998.

[93] R. Mateescu and H. Garavel. XTL: A meta-language and tool for temporal logic model-checking. In T. Margaria and B. Steffen, editors, *Proceedings of the International Workshop on Software Tools for Technology Transfer STTT'98 (Aalborg, Denmark)*, number NS-98-4 in BRICS Notes Series, pages 33–42, July 1998.

[94] R. Mateescu and M. Sighireanu. Efficient on-the-fly model-checking for regular alternation-free mu-calculus. In S. Gnesi, I. Schieferdecker, and A. Rennoch, editors, *Proceedings of the* 5^{th} *International Workshop on Formal Methods for Industrial Critical Systems FMICS'2000*, pages 65–86, April 2000.

[95] S. Mauw. *PSF — A Process Specification Formalism*. PhD thesis, Programming Research Group, University of Amsterdam, 1991.

[96] S. Mauw, M.A. Reniers, and T.A.C. Willemse. Message Sequence Charts in the software engineering process. In S.K. Chang, editor, *Handbook of Software Engineering and Knowledge Engineering*, volume I: Fundamentals, pages 437–463. World Scientific, 2001.

[97] R. Milner. *Communication and Concurrency*. Prentice Hall International, 1989.

[98] F. Moller and C. Tofts. A temporal calculus of communicating systems. In J.C.M. Baeten and J.W. Klop, editors, *Proceedings CONCUR'90*, volume 458 of *Lecture Notes in Computer Science*, pages 401–415. Springer-Verlag, 1990.

[99] X. Nicollin and J. Sifakis. The algebra of timed processes ATP: Theory and application. *Information and Computation*, 114(1):131–178, 1994.

[100] J. Ouaknine and J. Worrell. Universality and language inclusion for open and closed timed automata. In *Proceedings of the Sixth International Workshop on Hybrid Systems: Computation and Control (HSCC'03)*, Lecture Notes in Computer Science, 2003. To appear.

[101] G.D. Plotkin. A structural approach to operational semantics. Technical Report DIAMI FN-19, Computer Science Department, Aarhus University, 1981.

[102] A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesize. In *Proceedings of the* 31^{st} *IEEE Symposium on Foundations of Computer Science*, pages 746–757, October 1990.

[103] A. Pnueli, J. Xu, and L. Zuck. Liveness with (0,1,infinity)-counter abstraction. In E. Brinksma and K.G. Larsen, editors, *14th International Conference on Computer Aided Verification, CAV 2002*, volume 2404 of *Lecture Notes in Computer Science*, pages 107–122. Springer-Verlag, 2002.

[104] J. Quemada, D. de Frutos, and A. Azcorra. TIC: A TImed Calculus. *Formal Aspects of Computing*, 5(3):224–252, 1993.

[105] P.J. Ramadge and W.M. Wonham. Supervisory control of a class of discrete-event processes. *SIAM Journal of Control and Optimization*, 25:206–230, 1987.

[106] P.J. Ramadge and W.M. Wonham. The control of discrete event systems. In *Proceedings of the IEEE*, volume 77(1), pages 81–98, 1989.

[107] M. Raynal. *Algorithms for Mutual Exclusion*. North Oxford Academic, 1986.

[108] M.A. Reniers, J.F. Groote, M.B. van der Zwaag, and J. van Wamel. Completeness of timed μCRL. *Fundamenta Informaticae*, 50(3–4):361–402, 2002.

[109] K. Rudie and W.M. Wonham. Supervisory control of communicating processes. In L. Logrippo, R.L. Probert, and H. Ural, editors, *Protocol Specification, Testing and Verification X*, IFIP, pages 243–257. Elsevier Science Publishers B.V. (North-Holland), 1990.

[110] V. Rusu and E. Zinovieva. Analyzing automata with Presburger arithmetic and uninterpreted function symbols. In R. Mayr, editor, *Verification of Parameterized Systems - VEPAS 2001*, volume 50(4) of *Electronic Notes in Theoretical Computer Science*. Elsevier, 2001.

[111] C. Shankland. Using E-LOTOS to pick a leader. In *Proceedings of the Workshop on Formal Methods in Computation, Ullapool, UK*, pages 143–162, 1999.

[112] C. Shankland and M.B. van der Zwaag. The tree identify protocol of IEEE 1394 in μCRL. *Formal Aspects of Computing*, 10:509–531, 1998.

[113] M. Sighireanu. LOTOS NT user's manual (version 2.1). Technical report, INRIA Rhône-Alpes projet VASY, November 2000.

[114] M. Sighireanu and R. Mateescu. Validation of the link layer protocol of the IEEE-1394 serial bus ("firewire"): an experiment with E-LOTOS. In I. Lovrek, editor, *Proceedings of the 2nd COST 247 International Workshop on Applied Formal Methods in System Design*, June 1997.

[115] O. Strichman, S.A. Seshia, and R.E. Bryant. Deciding separation formulas with SAT. In *14th International Conference on Computer Aided Verification, CAV 2002*, volume 2404 of *Lecture Notes in Computer Science*, pages 209–222. Springer-Verlag, 2002.

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

[117] W. Thomas. On the synthesis of strategies in infinite games. In E.W. Mayr and C. Puech, editors, *Theoretical Aspects of Computer Science*, volume 900 of *Lecture Notes in Computer Science*, pages 1–13. Springer-Verlag, 1995.

[118] K.J. Turner. The invoice case study in (E-)LOTOS. In M. Allemand, C. Attiogbé, and H. Habrias, editors, *Proceedings of the International Workshop on Comparing System Specification Techniques (Nantes, France)*, March 1998.

[119] Y.S. Usenko. *Linearization in μCRL*. PhD thesis, Eindhoven University of Technology, December 2002.

[120] J.J. Vereijken. *Discrete-Time Process Algebra*. PhD thesis, Eindhoven University of Technology, December 1997.

[121] Y. Wang. Real-time behaviour of asynchronous agents. In J.C.M. Baeten and J.W. Klop, editors, *Proceedings CONCUR'90*, volume 458 of *Lecture Notes in Computer Science*, pages 502–520. Springer-Verlag, 1990.

[122] Y. Wang. *A Calculus of Real-Time Systems*. PhD thesis, Chalmers University og Technology, 1991.

[123] T.A.C. Willemse. The Analysis of a Conveyor Belt System: a Case Study in Hybrid Systems and Timed μCRL. Technical Report CSR 99-10, Eindhoven University of Technology, Department of Mathematics and Computing Science, 1999.

[124] T.A.C. Willemse. A Process Algebraic Approach to Hybrid Systems. In J.P. Veen, editor, *Workshop on Embedded Systems*, October 2000.

[125] T.A.C. Willemse. Interpretations of automata. Technical Report CSR 01-02, Eindhoven University of Technology, Department of Mathematics and Computer Science, 2001.

[126] A.G. Wouters. Manual for the μCRL tool set (version 2.8.2). Software Engineering Report SEN-R0130, CWI, 2001.

[127] M.B. van der Zwaag. *Models and Logics for Process Algebra*. PhD thesis, University of Amsterdam, October 2002.

Samenvatting

Formele methoden stellen gebruikers in staat om het ontwerp van systemen in een vroegtijdig stadium reeds te analyseren en te verbeteren. Het is echter niet zo dat dit ook tevens tot gevolg heeft dat formele methoden worden toegepast; het tegendeel lijkt eerder waar te zijn. Hiervoor worden vaak diverse redenen aangedragen. Formele methoden worden veelal gezien als duur: ook zonder gebruik te maken van formele methoden kunnen systemen gebouwd worden. De dagelijkse praktijk levert ons tal van voorbeelden van projecten die succesvol ten einde worden gebracht zonder het toepassen van formele methoden. De toegevoegde waarde van formele methoden is dus niet eenvoudig in te zien. Bovendien wordt vaak als reden aangevoerd dat formele methoden moeilijk zijn. Als laatste wordt vaak genoemd dat formele methoden maar beperkt toepasbaar zijn, en juist daar waar ze daadwerkelijk nodig zijn, niet kunnen worden gebruikt. Met name deze laatste twee punten van kritiek liggen ten grondslag aan het huidige onderzoek dat wordt verricht in formele methoden. Het zijn ook deze twee punten van kritiek waar dit proefschrift zich op richt.

In vijf hoofdstukken worden, gebruik makend van het formalisme μCRL, en haar uitbreiding met tijd μCRL_t, de volgende drie gebieden bestudeerd:

1. de analyse en specificatie van data-afhankelijke systemen. Door beschikbare technieken aan te scherpen en uit te breiden met nieuwe mogelijkheden, zullen formele methoden op een steeds groter vlak bruikbaar worden. Met name de techniek, bekend als *model-checking*, is relatief zeer gebruiksvriendelijk. Middels een programma kan men automatisch controleren of een systeem aan een —door de gebruiker geformuleerde— eigenschap voldoet. Het analyseren van een systeem is hierbij dus volledig geautomatiseerd. Hoofdstuk 3 van dit proefschrift beschrijft een uitbreiding van de model-checking techniek die ertoe bijdraagt dat een grotere klasse van systemen automatisch geanalyseerd kunnen worden. In hoofdstuk 4 wordt een techniek beschreven die ervoor zorgt dat het te analyseren systeem reeds beperkt wordt tot dàt gedeelte van het systeemgedrag dat überhaupt bereikt mag worden. Een dergelijke techniek kan systeemontwerpers een leidraad verschaffen voor het uiteindelijke systeemontwerp.

2. de overeenkomsten en verschillen tussen μCRL_t en de formalismen *timed automata* en *hybrid automata*. De twee laatstgenoemde formalismen zijn populaire formalismen om tijdsafhankelijke systemen te beschrijven en analyseren. Om tot een beter begrip te komen van zowel timed automata, hybrid automata en μCRL_t, is een vergelijking tussen deze formalismen noodzakelijk. De hoofdstukken 5 en 6 uit dit proefschrift bestuderen de onderlinge relaties tussen μCRL_t, timed automata en hybrid automata. Uit dit onderzoek blijkt dat de klasse van systemen die beschreven kan worden met behulp van timed automata een onderdeel vormen van de klasse van systemen die beschreven kunnen worden

met behulp van μCRL$_t$. Dit gaat ten dele ook op voor hybrid automata. We tonen eveneens aan dat μCRL$_t$ en hybrid automata geen overeenstemming bereiken over een basis concept als *time additivity*.

3. de volwassenheid van analyse technieken van μCRL$_t$. Het onderzoek naar de technieken die ontwikkeld zijn voor de analyse van systemen waarin tijd geen rol speelt, is inmiddels ver gevorderd. Zo kan op relatief eenvoudige wijze aangetoond worden, door gebruik te maken van abstractie, dat een *black-box* beschrijving van een systeem equivalent is aan een zekere *white-box* beschrijving van datzelfde systeem. Tijdsafhankelijke systemen zijn echter in zekere zin complexer, en de technieken die voor tijdsonafhankelijke systemen toe te passen zijn, zijn minder, of helemaal niet van toepassing voor dit soort systemen. Om te toetsen tot op welke hoogte μCRL$_t$ geschikte technieken in huis heeft om tijdsafhankelijke systemen te analyseren, onderzoeken we in hoofdstuk 7 hoe μCRL$_t$ toe te passen is op een complex voorbeeld. Hieruit blijkt dat slechts een deel van de gewenste analyse ondersteund wordt door de technieken van μCRL$_t$. Bovendien blijkt het dat daar waar μCRL$_t$ de technieken biedt om de analyse te verrichten, deze onpraktisch zijn. Voor eventuele verbeteringen op het gebied van de analyse technieken van μCRL$_t$ verwijzen we naar de slotopmerkingen van het desbetreffende hoofdstuk.

De behandelde onderwerpen geven aan dat een gedeelte van de kritiek op formele methoden wel degelijk gegrond is. Zo blijkt uit bijvoorbeeld hoofdstuk 7 dat de technieken voor de analyse van tijdsafhankelijke systemen in μCRL$_t$ moeilijk zijn toe te passen. Bovendien blijkt uit het feit dat de formalismen μCRL$_t$ en hybrid automata geen overeenstemming bereiken over basis concepten zoals *time additivity*, de theorie nog niet uitgekristalliseerd is. Echter, we constateren ook dat de technieken op het gebied van analyse van tijdsonafhankelijke systemen praktische vormen hebben aangenomen. In wezen vormen de technieken voor de laatstgenoemde systemen geen belemmering voor de toepassing van formele methoden in de praktijk.

Curriculum Vitae

10 januari 1974	geboren te Grubbenvorst
1986–1992	Atheneum B, St. Thomascollege te Venlo
juni 1992	Diploma VWO
1992–1998	Studie Technische Informatica, Technische Universiteit Eindhoven
1997–1998	Stage CMG-ATG, Den Haag
juni 1998	Diploma Technische Informatica *cum laude*
1998–2002	Assistent in Opleiding, Formele Methoden, Technische Universiteit Eindhoven
2002–nu	Post-doctoraal onderzoeker, Formele Methoden, Technische Universiteit Eindhoven

Titles in the IPA Dissertation Series