Controller Design from Temporal Logic: Undecidability Need Not Matter

Doctoral Dissertation
submitted to the
Technical Faculty of the
Christian-Albrechts-Universität
Kiel, Germany

submitted by

Martin Fränzle

Kiel, December 1996

Preface

When setting up new scientific theories, simplifying assumptions can be an indispensable tool for developing and debugging the general shape of a theory without getting involved with too much distracting detail. In later stages of the development, these simplifications should be resolved or at least their impact on the theory should be analyzed, as otherwise any deduction drawn from the theory might be an artefact arising from simplifications rather than a property of the problem domain. In general, not all simplifications can be resolved without destroying an otherwise useful theory. But within any mature engineering discipline we may at least demand that their impact on the engineering process has been analyzed with scrutiny, as otherwise the theory might be utterly useless to responsible engineers designing reliable systems.

In this respect, the design of real-time computer systems can be considered as an engineering discipline still in its infancy, as most of the underlying theory still refers to strongly idealized views of computation. Almost ubiquitous in this field is the view of computation steps as timeless actions hopping from one discrete state to another, and extremely frequent are overly idealized representations of clock devices, characterizing them as continuously growing or even globally synchronized. It has been a while since Carl Adam Petri warned the formal methods community in computer science that such theories might be a serious risk when put to engineering practice:

> "... you can specify what you wish. But the implementor who has to construct these [specified] delays cannot have equality between two separate technical items. He cannot have *equality*, he can only have *similarity*. So the question arises: Are these clocks [specified in a previous example via delays] really synchronous if they are specified by an equal sum of delay times? If you specify so, in your abstraction they are synchronous. In any implementation you must be afraid that they are not synchronous, and *I have seen many such implementations fail in practice.*"
>
> *C. A. Petri in* [Pet86].

Although this remark did not immediately provoke a wave of new research, it did not remain unheard. Among the research groups that did take the challenge where the ProCoS projects [B^+89, B^+93], which designed their intermediate languages such that assumptions on the accuracy of clock devices have to be explicitly stated and thus can be checked upon implementation [FMO93, FvK93].

Although a similar approach is nowadays becoming popular in the field of temporal logics, this particular line of research did not gain much interest in the formal methods community, particularly the theoretically inclined one, presumably because of its considerable loss of smoothness of concepts compared to their idealized counterparts. Thus, in the end of ProCoS, the question still stood whether there are formal models of real-time computation that are tractable with sufficient ease yet do not impede safe implementation through over-idealization. At that time, a significant blow-up of complexity seemed to be the price to be paid for avoiding over-idealization — a price that would further reduce the attraction of formal methods as a design paradigm for real-time systems.

This state of affairs has provoked the research reported in this thesis. We have been motivated by the belief that a rigorous analysis of real-time computation would uncover smoother models than those considered in the ProCoS projects. This analysis has become a long and winded track, as even low-level, automata-based models of real-time computation proved to be remarkably instable against small changes in their definition — an indication that in contrast to the remarkably stable qualitative-time models of computation they are not a mature representation of the really implemented computation paradigm. In search for stable models we had to go down to far lower levels than automata models. We ended up dealing with continuous behavioural models of VLSI and building sound abstractions to come back to discrete-state models. Amazingly enough, this long tour has reached a point we never expected to find: in the end, the design problem of embedded real-time controllers proved to be even simpler without the original idealizations, bringing forth new decision and synthesis procedures!

Having enjoyed this winded track for myself, I would now like to invite the reader to a guided tour to its vantage points.

Kiel, June 1997
Martin Fränzle

Acknowledgements

Research is a process that is always in risk of deadlock. It is only through external stimulation and support of the researcher that a certain amount of liveness can be enforced. Therefore, quite some persons and institutions deserve my sincerest thanks for supplying me with energy to be dissipated while doing work on my thesis.

Biochemically, like any human being, I took that energy from burning loads of food. Our culture offers a wide choice of suitable input to this chemical process, but only some of the available options are of low enough entropy to be pleasing to both the eye and the tongue. As my wife Barbara provided so many enjoyable forms of energy supply, the sincerest thanks for protecting my research from deadlock (or, indeed, from starvation) go to her. As in our society acquiring food requires money, a personal income becomes just another prerequisite of research. I am grateful to Prof. Dr. Hans Langmaack for freeing my mind and stomach for research by providing me with an employment and by always being busy obtaining continuations; Claudia Herbers is to be thanked for taking much of the organizational trouble connected with this. Funding for my research was supplied by the Commission of the European Union (under contracts EP 3104 and EP 7071), by the Deutsche Forschungsgemeinschaft (under grants DFG La 426/13-1 and DFG La 426/13-2), and by the state government of Schleswig-Holstein.

If a research activity is not compromised by an instable energy supply, psychological factors become important, as the energy can be dissipated on lots of other things than a thesis. This inevitably happens, as most other things become extremely interesting while you are working on the thesis. It needs a cosy home for relaxing, a partner being really interested in your work, and good friends to bring you back to what you are expected to do. I would like to thank my wife for providing that atmosphere and for relaxing me from so many tasks that I originally promised to take, and my thanks go to my daughter Mirjam for allowing me to go to office although being convinced that money for living comes from the cash automaton, not from my job. Furthermore, I am grateful to my parents for providing me with support in any respect and to my friends, particularly those at the students' parish, for helping me ease my mind whenever my research entered one of its numerous blind alleys.

Last-not-least, research is a distributed process. Hence, even if there is a sufficient share of calories and affection for doing it, research can become deadlocked by the unwillingness of other researchers in the field to interact. Fortunately, this never happened to me. It was Prof. Hans Langmaack who introduced me to the field of semantics, formal methods, and verification, and it was through his initiative and his confidence in his students' skills that I got international contacts within the ProCoS project even before doing my master's thesis. From then on, I benefited a lot from exchange with other researchers. My ideas on refinement put forward in this thesis have been largely influenced by Tony Hoare (Oxford), Ernst-Rüdiger Olderog (Oldenburg), Anders P. Ravn (Lyngby), and many other researchers of the ProCoS project. Testing preorders as a general notion of refinement have been subject of mail exchange with Søren B. Lassen (Aarhus) and dynamic models of VLSI behaviour have been discussed with Mark Greenstreet (Vancouver). Ben Moszkowski (Newcastle) shared his insights in interval temporal logic and hardware models. Michael R. Hansen (Lyngby), Paritosh Pandya (Bombay), and Manoranjan Satpathy (Macau) commented on my decidability and synthesis results concerning Duration Calculus. Prof. Hans Langmaack and my colleagues Bettina Buth, Karl-Heinz Buth, Burghard von Karger, Markus Müller-Olm, and Ruben-Benjamin Reincke at Kiel were always open to follow the evolution of my ideas and to comment on it. Among those, Markus Müller-Olm has had the predominant role in making this process really an evolution rather than mere mutation. Not only that he (and his stomach) stood sharing a room and some hectolitres of coffee with me during the past few years. It was also him who was always willing to filter my fuzzy ideas in search for their (often minor) solid core. Whenever you find a crisp condensation of an idea in this thesis it is not unlikely that some time ago Markus has had a hard time following an unaccessible presentation of that same idea. I will gratefully remember this support.

Finally, even if research comes to a successful outcome, its success is not necessarily perceived as such. During its evolution, science has become scattered into a universe of disciplines and fields, and research crossing some of those boundaries may encounter problems in finding its audience. Therefore, I am grateful to Hans Langmaack, Wolfgang Thomas, and Bengt Jonsson for their willingness to promote my ideas through refereeing and defense.

Contents

1

Introduction

Due to rapidly dropping costs and rapidly increasing power and flexibility of embedded digital hardware, digital control becomes more and more extensive in systems encountered in everyday life. Modern means of transport rely on digital hardware even in vital sub-systems like anti-locking brakes, fly-by-wire systems, or signaling hardware. Medical equipment can gain such a boost in functionality from exploiting the flexibility of computer control that more classical technology is no longer prevalent even in such critical applications as life-support systems or radiation treatment. And all kinds of financial transactions do nowadays rely on the availability of computer networks. It is obvious that digital systems, including custom hardware and software components, become crucial to the safety of human life and property.

A consequence of the desired operational flexibility of customized hardware and software is that controlling or even certifying its quality is an inherently complex task. Therefore, different aspects of quality have been identified that can be treated almost separately through loosely correlated measures. These aspects include *ergonomics*, which is concerned with providing the operator with a well-suited interface, *security*, which is prevention of unauthorized access to sensitive data and applications, and *safety*, which is concerned with guaranteeing the desired functionality. Measures concerning safety include *fault-tolerance*, i.e. methods for maintaining a possibly reduced level of functionality despite brake-down of system-components, as well as all actions towards reducing the risk of error in specification, design, and implementation of systems. The latter goal has given birth to a variety of guidelines, development models, and programming concepts, but also to a field of computer science called *formal methods*, being devoted to mathematical support of the design steps.

The idea of formal methods is to provide unambiguously defined languages for different design levels encountered during system development such that design steps can be justified either through rigorous or even formal proof or through "calculation" of the lower-level representation from its high-level counterpart. Focusing on the main computer applications at the time of their development, the first successful formal methods aimed at sequential programs. Among the most prominent formalisms for those systems are VDM [BJ78, Jon86], Z [Spi89], and B [ALN$^+$91]. Later, perhaps in correlation with the growing importance of computer control, formalisms dedicated to concurrent and reactive systems emerged. Those come along two mainstreams, which are temporal logics (situation-based[1] as in e.g. [MP92] or interval-based [Mos85]) and process calculi (e.g. CSP [Hoa85] and CCS [Mil89]). These formalisms treat time qualitatively, abstracting from the metric aspects of time. Naturally, this proved to be insufficient for expressing some of the safety aspects connected to computer control. Within transportation systems or medical equipment, but even within some of the protocols involved in financial transactions, some of the safety risks occur if and only if the environment may evolve for a certain period without interaction of the control system. This has lead to extensions of the above formalisms with (dense) metric time, among them Temporal Logic of Age [MP93a] as a situation-based dense-time temporal logic, Duration Calculus [ZHR91] as an interval-based dense-time temporal logic, and Timed CSP [DS89] as a process calculus. In all these cases, the basic pattern of the extension is similar, although this correspondence becomes somewhat blurred by technicalities involved: the non-negative reals are taken as a time domain, and state changes are tagged by a time stamp from the time domain. Then, the basic (qualitative time) formalism is extended by operators that can control the allowable passage of time between actions (in the case of Timed CSP), between changes of the truth value of a predicate (in the case of the Temporal Logic of Age), or even accumulated over phases that a predicate holds true (for Duration Calculus).

These extensions to dense metric time are intuitively appealing due to the clear underlying mathematical idea.[2] Nevertheless, we feel that this is more an argument resorting to mere mathematical beauty than a justification of a design instrument, which a formal method should be in the light of the aforementioned. The dense-time formalisms could fail to be design instruments in two respects: on one hand, they could provide an *unsound* image of the application domain, as no embedded controller can do a discrete state change in a time instant, which is apparently postulated by tagging state changes with real-valued time stamps. The consequence of such an unsound representation of the dynamics of embedded control hardware could be that systems developed within the formal model are not even implementable by embedded controllers. In this respect, the models might be *too coarse*, and it might be advisable not to tag state changes with time instants, but to even represent the trajectories encountered during state changes. On the other hand, the representation of embedded control by time-stamped state changes could equally well be *overly detailed*, distinguishing trajectories of the controller that will have the same impact on any controlled system, as no controlled system would be able to ascribe an exact time instant to a state change of the controller. To this end, dense-time formalisms would provide a sound, but overly cumbersome design model, branding perfectly well-suited designs as incorrect. We will adopt the well-known terminology used for logical calculi and will call such a design model *incomplete*.

Both problems sketched in the previous paragraph can be illustrated by means of an example from everyday life: the problem of designing safe traffic light systems. Obviously, one of the key issues in developing such a system is to ensure that certain combinations

[1]Indeed, situation-based temporal logics come in yet again two brands, viz. linear time and branching time, but we are not aiming at completeness of our list of examples.

[2]That is not to say that the extensions are mathematically simple. Indeed, there is a lot of hard technicality involved, particularly with the branching time-structure of Timed CSP.

of green and yellow lights do never occur, namely those that allow traffic on conflicting lanes to move simultaneously. Now assume that in the design phase a model assigning time *instants* to switching events is used. I.e., in analogy to the dense-time models discussed earlier, switching a bulb on or off is considered to be an atomic event that does not take time. Using such a model, we can design a traffic light controller and may even verify that it does meet the above requirement of not simultaneously lighting yellow and green bulbs on conflicting lanes. Yet, light bulbs need considerable time for getting dim after they have been switched off. Therefore, a photograph of the implemented crossing could easily reveal that sometimes — although only in transients — combinations of lights occur that we have *verified* to be absent.[3] I.e., the design model used is an *unsound* representation of the design problem, as it allows deduction of observably untrue properties.

On the other hand, one would not normally care about the delayed reaction of light bulbs when designing a traffic light system. The reason is that the reaction times of the human visual system and motoric functions are even larger such that these short transients of unsafe signaling cannot have impact on the controlled system, namely the flow of traffic. That is, although they are observable, they are only observable in a way that has no impact on the safety of the overall systems. Hence, the problem of inertia of the light bulbs should not be considered harmful and, furthermore, even little deviations in switching times will be irrelevant to overall safety due to the large inertia of the controlled system. Hence, implementation technologies for the traffic light controller are admissible even if they fail to perfectly implement the switching times that the formal design, which assigns time instants to switching events, requires. Small deviations e.g. due to mechanic switches with slightly unpredictable switching performance will not influence safety. To this end, the formal design model is *incomplete* as it might refuse such implementations due to possible temporal reversion introduced into the switching sequence. That is, through taking into account just those kinds of observations on the traffic lights that are relevant to the safety of the overall system, we can not only cure the unsoundness of the formal models that becomes apparent with respect to stronger notions of observability (e.g. taking photographs), but we can also show that the model can be weakened even further without risking *unsoundness within the application domain*.

It should be clear from the above that there is need to analyze the apparently straightforward introduction of metric time into reactive systems formalisms with respect to both its soundness and its completeness. As the traffic-light example indicates, answering these questions might, although they are formal antipodes, take advantage of the same mechanism. In the example, the same argument that allowed us to understand why the formal model — despite observable deficiencies — is sound *in its application domain* (we might also say fit-for-purpose) lead us to an idea of how to even increase completeness. This thesis is devoted to such an analysis for embedded real-time control, and indeed follows the line of the example in that it analyses the safety requirements that might be put on a complete

[3] In general, the requirement will not simply be that conflicting signals are not simultaneously lit, but even that there is a minimum delay between switching off the first light bulb and switching on the other one. Yet, this does not invalidate our argument, as then two consecutive photographs may show that the required delay is not respected.

plant in order to figure out how fine-grain the description of embedded controller dynamics has to be.

At this point, the reader might ask whether there is any evidence for similar soundness or completeness problems with current formal models of embedded control. Some evidence can be gained from observing the state of the art, as manifested in case studies of formal design of embedded controllers: when studying major comparative case studies, like the Karlsruhe production cell [LL95] or the Dagstuhl steam-boiler [ABL96], it is stunning that while very sophisticated analysis and refinement steps are performed at specification level, the actual implementation (i.e., a control program in some executable language) is hardly ever formally derived or verified. Instead, control programs are normally torn out-of-the-hat once the specification is felt to be close enough to a program. Obviously, a fundamental *semantic gap* between the applied formalisms and programming languages is perceived in this stage, as otherwise formal refinement or verification should be straightforward once the specification comes close to a program.

This sad state of affairs could nevertheless be an indication of just lacking maturity of the authors in dealing with their formalisms. But a recent discovery of the author, to be fully explained in chapter 2 of this thesis, provides hard technical evidence of the problem: a major number of undecidability results for real-time formalisms was found to lack physical interpretation in the field of embedded control, i.e. within just that application domain that these formalisms are deemed to describe. The reasons for the lacking physical interpretation are akin to the minor importance of bulb lighting times or small switching inaccuracies in the traffic-light example in that they are observability constraints, albeit more universal ones: any embedded control system is connected to the controlled plant through a finite number of sensors and actuators, which are clearly channels of finite capacity in the information-theoretic sense. Yet the proofs of those undecidability results rely on encoding infinite information in compact Cartesian products of state and time, whereas only a finite part thereof can become effective in any embedded control application due to the finite capacity of sensor and actuator channels. This means that these undecidability results depend on properties of the controller which are not observable by the environment in any embedded control application. Such undecidability results, as remarked in the title of this thesis, need not matter to the design problem of embedded controllers.

There are thus strong indications of completeness problems of existing real-time formalisms with respect to the design problems they are intended to cover. However, this argument only evidences possible shortcomings of the formalisms rather than shows possible remedies. The major part of this thesis is thus devoted to exemplifying how these insights can be put to advantage for setting up new decision and synthesis procedures. We have selected a sample real-time formalism for this. It is Duration Logic, which corresponds to the core of the Duration Calculi [Zho93], a recently developed major line of real-time formalisms.

Structure of the thesis

We start our exploration with a demonstration of the claimed divergence between formal methods of embedded control and the problem of designing embedded controllers: chapter 2 is concerned with the relevance of undecidability of real-time formalisms to the availability of effective controller verification and synthesis procedures. Common proof patterns for undecidability of real-time formalisms are opposed with information-theoretic constraints encountered in embedded control applications. This analysis reveals that a major number of undecidability results for real-time and hybrid-systems formalisms encodes more information in individual trajectories than can become effective in any embedded control application due to the finite capacity of sensor and actuator channels. We conclude that some of the intricacy of designing embedded controllers within formal methods might be an artefact of the formalisms rather than an inherent complexity of the design task.

The remainder of this thesis aims at exploiting this observation on Duration Logic, and is therefore divided into three parts. Part I, comprising chapters 3 to 7, is concerned with introducing Duration Logic as a formalism for describing real-time behaviour and analyzing its decidability properties. Part II, which contains chapters 8 and 9, sets up a synthesis procedure for synchronous controllers and applies it to the ProCoS gas-burner case-study [RRH93], while part III deals with automatic verification of asynchronous circuits.

In **part I**, chapters 3 and 4 prepare for a mathematical treatment of real-time embedded control by introducing some general mathematical notation and formalizing trajectories over time as well as timed regular expressions as basic models of real-time behaviour. Chapter 5 concentrates on defining Duration Logic, the sample formalism used throughout the thesis for exemplifying exploitation of our hypothesis concerning the irrelevance of certain undecidability results to embedded control.

Chapter 6 is devoted to decidability issues of Duration Logic and some other subsets of the original Duration Calculus [ZHR91]. While undecidability of the validity problem of Duration Logic on finitely variable trajectories is known from the literature [ZHS93], we will show that the validity problem of Duration Logic is even undecidable if the logics is interpreted by trajectories that have at most a fixed number of discontinuities in any unit interval of time. This is in contrast to e.g. language inclusion problems of timed automata or the validity problem of the monadic second-order logic of temporal distance, which become decidable if interpretation is restricted to such trajectories [Wil94b]. This witnesses that one indeed needs special techniques to obtain automatic controller verification or synthesis procedures for Duration Logic: restriction to trajectories having at most a fixed number of discontinuities in any unit interval of time is a notion of band-limitedness, which is easily justifiable from physical principles of embedded control, but unfortunately even that is insufficient for making Duration Logic decidable.

However, there are even more restricted model classes on which Duration Logic becomes decidable. Still in chapter 6, we show that Duration Logic is decidable on time-wise discrete trajectories, which are trajectories where state changes occur only in integer time instants. For the first time ever, this provides a positive decidability result for a true Duration Calculus with unrestricted negation on a dense time domain.[4] But this decidability result for

[4]Given the discrete nature of time-wise discrete trajectories, the reader may wonder why we call this result a decidability result on a dense time domain. However, even despite the restricted model class, the temporal operators of Duration Logic still quantify over a dense time domain in this interpretation. Hence, this is still a dense-time interpretation of Duration Logic, and until recently, its decidability was unclear.

Duration Logic depends on the drastic restriction of the model class to time-wise discrete trajectories. As embedded controllers are usually subject to unsynchronized inputs, this model class does not even cover their possible behaviours, such that this decidability result alone is insufficient for overcoming the lack of automatic controller verification and synthesis procedures based on Duration Calculi.

Chapter 7 is concerned with the feasibility of effective controller verification and synthesis procedures based on Duration Logic. It starts with a review of a common argument why undecidability of the validity problem of a linear-time temporal logic on finitely variable trajectories implies impossibility of such automatic procedures. While this seems to imply that automatic controller verification and synthesis from Duration Logic is impossible, we reveal that this argument is flawed by the problem observed in chapter 2: it relies on an undecidability result that encodes more information in individual trajectories than can become effective in any embedded control application due to the finite capacity of sensor and actuator channels. We conclude that automatic verification and synthesis procedures for Duration Logic may exist.

Parts II and III demonstrate that this is indeed the case: **Part II** deals with synthesis of synchronous finite-state controllers from control problems formulated in Duration Logic. Chapter 8 reveals such a synthesis procedure through formalization and analysis of switching latencies of sensors. As far as we know, this is the first synthesis procedure that was ever proposed for a dense-time Duration Calculus. Chapter 9 describes an application of this synthesis procedure to the ProCoS gas-burner case study [RRH93], witnessing applicability of the algorithm.

Part III explores automatic verification of asynchronous circuits. Before getting involved with circuits, chapter 10 deals with the general principles by which reactivity constraints (i.e., limits to information-flow) of system components can lead to a simplification of the system verification problem. Based on the insight that one system (e.g., a controller) may safely replace another one (even an abstract representation, i.e. a specification) if there is no application context in which the former invalidates an expectation on the complete system that the latter satisfies, a generalization of test preorders [Hen88] is identified as a natural notion of refinement in embedded systems. Proof rules for this notion of refinement are given.

Chapter 11 prepares for an application of these ideas: it is concerned with introducing a suitable meta-language for such a refinement proof dealing with both discrete and continuous system states. Therefore, Duration Logic is extended to a simple formalism for describing so-called hybrid systems, i.e. systems consisting of a non-trivial mixture of discrete and continuous components.

In chapter 12, the machinery made available in the previous two chapters is exploited for formally verifying correctness of a discrete-state model of digital hardware with respect to a continuous, circuit-level model of its implementation by transistor networks. This closes, on a prototypic example, the semantic gap between the discrete-state models used in hardware design (and verification!) and the actual switching dynamics of the implemented hardware. As this gap between discrete and continuous models of hardware behaviour has rarely been attacked by formal methods, we take this result as a witness that our notion of refinement can bridge some of the frequently perceived semantic gaps that otherwise constrain the scope of formal methods.

Chapter 13 analyzes the discrete-state hardware model of chapter 12 for the possibility of automatically model-checking it against formulae from a certain fragment of Duration Logic. While the immediate decidability result is negative, observability constraints can be exploited for deriving a more deterministic model that has a decidable satisfaction problem. The most interesting point about this construction is how observability constraints are used for obtaining a more tractable model: the latter model is derived from the former by simply *postulating* additional constraints on the dynamic behaviour. It is then shown that there is no context in which these postulates are observably untrue. Thus, decidability can be gained for free: there is no context in which the additional postulates allow unsound deductions concerning satisfaction of requirements by complete systems, yet they yield decidability of the controller correctness problem.

Chapter 14 provides a summary of the results of the thesis, combined with remarks on future research and possible fields of exploitation of the theories developed.

Furthermore, an appendix is supplied. It provides a different perspective on the undecidability of Duration Logic by showing that Duration Logic is closely connected to timed transition tables with stop-watches [HKPV95]. This appendix is intended to help readers that are more familiar with the automata-theoretic approach to real-time in assessing the expressiveness of Duration Logic.

2

Why Certain Undecidability Results Need Not Matter

The primary objective of studying formal models of real-time computation is to support the design process of real-time reactive systems. The main contribution that formal methods claim to make to this engineering discipline is verifiability. This requires semantic models that are defined with scrutiny in order to ensure their correspondence to the modelled entities, but it is becoming increasingly obvious that it also requires some sort of tool support for the verification tasks arising. Various kinds of tools have been investigated, ranging from proof checkers over user-directed theorem provers to fully automatic proof procedures. At least in the short run, the latter seem to have the best prospects for integration into existing engineering contexts. Therefore, the possibility of such 'key-press verification' techniques has become one of the first properties to be checked when setting up a new specification formalism.

The techniques to be used for such an investigation are classical within mathematical logic and automata theory: either a decision procedure has to be set up for the desired correctness property, which yields a prototype of the desired key-press verification method, or the correctness property has to be shown to be undecidable, which shows that key-press verification is in general impossible. Unfortunately, the results of such analyses have been negative for a lot of embedded-system design formalisms of appealing expressiveness. As prototypic examples for the vast number of such results we name just two:

1. The question whether a hybrid automaton, i.e. a combination of a finite-state embedded controller and a continuous plant, reaches an undesirable state is undecidable except for simplistic subclasses of the class of hybrid automata [HKPV95].

2. Language inclusion between the languages accepted by two timed automata is undecidable [AD94].

While these particular formal properties bear a strong automata-theoretic flavour, they are deemed to describe inherent complexity of the problem of designing reactive real-time

An earlier version of this chapter has been published as opening passage of [Frä96b].

controllers: property 1 is taken to say that it is undecidable whether an embedded finite-state controller may drive a plant into an undesirable situation. Property 2 is interpreted as a formal statement that even if the desirable controller behaviours can be formalized by a timed automaton there is no chance to automatically check whether a controller (taken to be some kind of timed automaton) conforms to them.

The correspondence between these pragmatic interpretations and the formal properties 1 and 2 seems rather straightforward. Therefore, they are generally accepted as rigorous arguments against the possibility of automatically verifying the corresponding correctness properties of embedded controllers. However, it is illuminating to see how properties 1 and 2 are verified, in particular to check whether the proofs have a physical interpretation in the field of embedded control. In the next section we thus take a philosophic perspective on the problem of embedded-controller design, confronting typical techniques used when proving undecidability of real-time and hybrid-systems formalisms with information-theoretic constraints arising in embedded control. This provides evidence that a major number of the proofs of undecidability results for real-time and hybrid-systems formalisms exploits artefacts of the formalisms rather than encodes properties of the problem domain, thus providing no physical interpretation of the undecidability results. This observation is central to this thesis as it can be exploited for development of automatic verification and synthesis procedures for reactive controllers from specification formalisms that currently seem to be out-of-scope of such procedures.

2.1 Common patterns of undecidability proofs

For obvious reasons, most undecidability proofs for real-time logics as well as reachability problems in integration graphs or hybrid automata follow the approach of encoding well-known undecidable problems within the formalism under investigation. Two frequently encountered proof patterns use encodings of two-counter machines, thus reducing undecidability of the investigated problem to undecidability of the halting-problem for two-counter machines.

The first proof pattern is the one supporting property 1 above. It is particularly illustrative for the philosophical problems encountered and, furthermore, it is widely used for showing undecidability of reachability in hybrid automata, or, as special cases, automata with multiple-slope clocks (e.g. in [HKPV95]).

First we note that, pragmatically, *hybrid automata* are studied as an idealized — and thus more well-suited to formal analysis — model of coupled digital and analog systems. Roughly, hybrid automata can be characterized as *finite automata* with transitions that are triggered by predicates on the (continuous) plant state and time, *combined with* a description of the *evolution of the continuous plant*. The latter consists of a set of real-valued variables that are governed by sets of (syntactically restricted) differential equations, from which a currently active set is selected depending on the current automaton state.

Proofs that for a given class of hybrid automata it is in general undecidable whether an automaton may reach a given state are frequently incarnations of

Proof pattern 1. *Encode a two-counter machine by a hybrid automaton (automaton with stop-watches, or automaton with 2-slope clocks, respectively), representing the counter values by two continuous variables of bounded range. E.g., by variables of range* $[0,1]$ *through the embedding* ε *defined by* $\varepsilon(k) = 2^{-k}$.

Although the results obtained are formally correct and absolutely well-done, we feel that they are of little relevance to the practical design problems hybrid automata are intended to cover. The encodings used (e.g. ε) encode infinite information, namely the set of natural numbers, within a compact interval of continuous states, whereas the ubiquity of noise limits the information content encodable by any bounded continuous variable encountered in real hybrid systems to a finite value. Hence, on simple information-theoretic grounds, the undecidability results thus obtained can be said to be artefacts of an overly idealized formalization.

When dealing with pure real-time formalisms, where continuous variables are absent and thus another encoding is used, we have to refine our argument a little bit. To exemplify this, we do now turn to the second proof pattern, which is frequently used for showing undecidability of real-time logics or finite-state automata models over dense time, e.g. for undecidability of dense-time Duration Calculus in [ZHS93, Theorem 11] or for undecidability of universality of the language accepted by a timed automaton in [AD94, Theorem 5.2].

Proof pattern 2. *Encode the counters of a two-counter machine through a corresponding number of state changes of discrete variables*[1] *within time-intervals of some fixed length* k.

This time, the state space is discrete and hence our previous argument of why the proof relies on artefacts of the formalism does not apply here. Nevertheless, we may note the following observation:

Both undecidability results rely on encoding infinite information within compact Cartesian products of state and time.

On the other hand, the general architecture of embedded control systems is as depicted in figure 2.1 on the following page, where sensors and actuators form *channels of finite capacity* [Sha49]. This means that within any compact interval of time only finite information about the plant state can be gathered by the controller and only finite information about the history of the controller can become effective to the plant behaviour. I.e., from the plant perspective there is only a finite number of distinguishable controller trajectories over any bounded interval of time such that *a physically observable encoding of infinite information requires unbounded time.* Therefore, undecidability results relying on encoding infinite information within finite time-intervals are lacking physical interpretation within the context of embedded control. Consequently, the relevance of undecidability results obtained through one of above two proof patterns to the possibility of key-press verification techniques or automatic synthesis is doubtful.

[1] In [ZHS93, Theorem 11], the discrete variables are two time-dependent Boolean variables, in [AD94, Theorem 5.2] it is the trace.

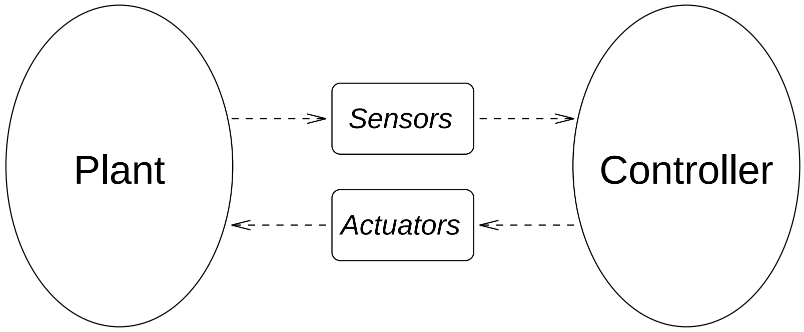


Figure 2.1: General architecture of embedded control systems

2.2 How to exploit the missing physical interpretation

Given that most undecidability results in the field of dense real-time formalisms are obtained through above two proof patterns, it is an interesting observation that such undecidability results may lack physical interpretation in the field of embedded control: this observation implies that fully automatic controller verification or synthesis procedures may exist even despite such undecidability results.

However, without a successful example this would remain just a hypothesis. Therefore, the remainder of this thesis is devoted to an exploitation of this observation on a sample real-time formalism, thereby obtaining new automatic synthesis and verification procedures. The key to putting the observations to use is evident from figure 2.1: it is formalization and exploitation of the reactivity constraints that interfaces impose on embedded control. We will use this technique for setting up a synthesis procedure for synchronous controllers from Duration Logic, the core of Duration Calculus [ZHR91], in part II and an automatic verification procedure for asynchronous circuits with respect to a subset of Duration Logic in part III of this thesis. However, we have to introduce Duration Logic and to analyze its decidability beforehand, which is the theme of part I.

Part I

Formal Specification of Timed Behaviour

Duration Logic, the core of Duration Calculus [ZHR91], is introduced and its decidability as well as decidability of certain other descendants of Duration Calculus is examined. We review a common argument why the established undecidability results indicate that key-press verification as well as automatic synthesis of embedded controllers from Duration Logic is impossible, yet provide philosophical evidence against this argument.

3

Mathematical Notation

This chapter lists, for the reader's convenience, some of the notational conventions that are used throughout this thesis within mathematical formulae. Except for section 3.4, which deals with properties that hold almost everywhere on subsets of the real numbers, this notation is quite standard and the exposition serves the purpose of a glossary that can be consulted whenever ambiguities might occur.

3.1 Relations

Most relations encountered will be binary, which allows infix notation. Hence, we will frequently write xRy as a convenient shorthand for $(x,y) \in R$. Furthermore, for binary relations (and functions in particular) it makes sense to talk of their *domain* and *image*. For a binary relation R these are denoted by

$$\begin{aligned} \mathrm{dom}\,(R) &\overset{\mathrm{def}}{=} \{x \mid \exists\, y\,.\, xRy\,\} \quad \text{and} \\ \mathrm{im}\,(R) &\overset{\mathrm{def}}{=} \{y \mid \exists\, x\,.\, xRy\,\}\ . \end{aligned}$$

3.2 Functions

Unless otherwise noted, all functions are total. Hence, for two sets A and B, $A \to B$ denotes the set of *total* functions from A to B and $f : A \to B$ denotes the fact that $f \in A \to B$. The set of partial functions from A to B is represented by $A \xrightarrow{\text{partial}} B$ and a partial function f from A to B by $f : A \xrightarrow{\text{partial}} B$.

It is often handy to have a functional abstraction operator. We use the arrow '$\mapsto$' and denote by $x : A \mapsto t$ the (in general) partial function

$$(x : A \mapsto t)(a) \overset{\mathrm{def}}{=} \begin{cases} t\{a/x\} & \text{iff } t\{a/x\} \text{ is defined,} \\ \text{undefined} & \text{iff } t\{a/x\} \text{ is undefined,} \end{cases}$$

from A to the range of values of t, where $t\{a/x\}$ denotes substitution of a for x in term t. If the argument set A is clear from context we simply write $x \mapsto t$ instead of $x : A \mapsto t$.

Furthermore, functions can be constructed from other functions using superposition $\oplus$ and functional composition $\circ$. Superposition $\oplus$ is defined by

$$(f \oplus g)(a) \stackrel{\text{def}}{=} \begin{cases} f(a) & \text{iff } g(a) \text{ is undefined,} \\ g(a) & \text{iff } g(a) \text{ is defined} \end{cases}$$

and yields a function $f \oplus g : A \cup B \xrightarrow{\text{partial}} C$ from two functions $f : A \xrightarrow{\text{partial}} C$ and $g : B \xrightarrow{\text{partial}} C$. For functional composition $\circ$ we follow the usual convention of writing down the functions to be composed in reverse order of application, i.e. $f \circ g \stackrel{\text{def}}{=} x \mapsto f(g(x))$. Finally, restriction of the domain of a function $f : A \xrightarrow{\text{partial}} B$ to a subset $A' \subset A$ is denoted by $f\big|_{A'} \stackrel{\text{def}}{=} x : A' \mapsto f(x)$.

3.3 Strings

The set of finite strings over an alphabet α, denoted α^*, is the set of partial functions from the natural numbers $\mathbb{N}$ to α which have a lower cut of $\mathbb{N}$ as domain. Note that string indices start with zero as $0 \in \mathbb{N}$. A string s with $\mathrm{dom}\,(s) = \mathbb{N}_{<k} = \{n \in \mathbb{N} \mid n < k\}$ is said to have length k, which is denoted by $\mathrm{len}(s) = k$. Concatenation of strings s and r is denoted by $s \cdot r$ and substring selection by

$$s[n, m] \stackrel{\text{def}}{=} k : \mathbb{N}_{<m-n} \mapsto s(k + n)$$

for a string s and natural numbers n and m. Note that substring selection is well-defined even if n or m are not viable indices in s, i.e. if $m \geq \mathrm{len}(s)$ or even $n \geq \mathrm{len}(s)$, and that the selected substring includes position n of s (if existent), but not position m.

3.4 Almost universal properties

Besides universal and existential properties, we frequently need to express properties that hold almost everywhere on the real numbers or a measurable subset thereof. For this behalf we introduce a special quantifier $\overset{\circ}{\forall}\ x \in M \,.\, \phi$ denoting the fact that ϕ holds almost everywhere on $M \subseteq \mathbb{R}$.

Definition 3.1. $\overset{\circ}{\forall}\ x \in M \,.\, \phi$, *where* $M \subseteq \mathbb{R}$, *holds iff* $\{x \in M \mid \neg\phi\}$ *is a set of measure zero.*

A derived concept is equality of functions almost everywhere, denoted $\approx$. Let f and g be (partial or total) functions from the reals into some set X. Then

Definition 3.2. $f \approx g$ *iff* $\mathrm{dom}\,(f) = \mathrm{dom}\,(g)$ *and* $\overset{\circ}{\forall}\ x \in \mathrm{dom}\,(f) \,.\, \big(f(x) = g(x)\big)$.

4

Fundamental Mathematical Models of Real-Time Systems

Formal methods in computer science refer to mathematical models of the objects under investigation in order to apply mathematical analysis techniques to the development of computerized systems. Being interested in embedded real-time control, we need a model of *dynamic systems* evolving dynamically over metric time, its independent variable. Mostly, we will adopt one of the oldest and best-studied models of such systems, used throughout in classical physics since Newton's seminal work and inherited by dynamic systems theory (e.g. [Lue79]). It is the model of *trajectories* over time, associating system *states* to time instants. Application of such a model requires two steps: first, selection of an appropriate time domain over which system dynamics is to be recorded. Second, identification of a suitable set of system states that describe observations that may be *conceptually* made of the system in such a time instant.[1] Being interested in embedded computer control of physical environments and thus in a smooth interface to convenient descriptions of environment dynamics we will adopt a time domain on which these descriptions can be directly interpreted, using connected subsets of the real numbers. As state spaces we may use products over both finite sets and connected subsets of the reals, the former modelling states of discrete components while the latter model continuous variables of the system. The particular appeal of this model comes from its close connection to physical modeling techniques, which eases interfacing to plant descriptions. Furthermore, as an aid to intuition, this allows even oscilloscope-like graphical representations of real-time behaviours, given that the state space is graphically representable on one axis such that the whole trajectory can be depicted on the plane as a graph over time.

The next sections are devoted to formally introducing the concepts of trajectories, finite observations on them, and timed regular expressions as an effective means for describing certain sets of finite observations over time.

[1] Note that properties ascribed to time instants are frequently only conceptual observations. Taking observations requires interactions that will generally take time. Hence, point properties on a sufficiently dense time domain are hardly directly observable, but can nevertheless be defined through e.g. limit properties, rendering them *conceptual* observations.

4.1 Time

For reasons of interfacing to physical models of environments and to control theory, which is an obvious prerequisite for any successful theory of embedded computer control, the time domain used throughout this thesis is the set of non-negative reals $\mathbb{R}_{\geq 0} \stackrel{\text{def}}{=} \{x \in \mathbb{R} \mid x \geq 0\}$. This particular instance of the real numbers will be denoted *Time*, with typical elements t and t'.

Being interested in system dynamics, we will be interested in certain connected subsets of *Time*, i.e. in certain intervals of time. The particular interval types are

- *Observation intervals*, which are nonempty closed and bounded intervals of time, i.e. intervals of the form $[a, b] \subset \mathit{Time}$ with $a, b \in \mathit{Time}$, $a \leq b$. The set of observation intervals is denoted *TimeInterval*.

- *Prefix intervals* which are observation intervals starting in time instant 0, i.e. nonempty intervals $[0, b] \subset \mathit{Time}$. The set of prefix intervals is denoted *PreInterval*.

- *Possibly unbounded intervals* $\mathit{TimeInterval}^{+} \stackrel{\text{def}}{=} \mathit{TimeInterval} \cup \{[a, \infty) \mid a \in \mathit{Time}\}$.

4.2 State spaces

When describing dynamic systems, a set of possible states the system may encounter is usually identified, which is called its *state space*.[2] We do not impose restrictions on the state spaces that may be used. However, the state spaces we encounter in this thesis are all of the forms $\Sigma = V \rightarrow \mathbb{B}$ and $\Sigma = \left\{\sigma : V_d \cup V_c \rightarrow \mathbb{B} \cup \mathbb{R} \,\middle|\, \mathrm{im}\left(\sigma|_{V_d}\right) \subseteq \mathbb{B} \wedge \mathrm{im}\left(\sigma|_{V_c}\right) \subseteq \mathbb{R}\right\}$, where V, V_d, and V_c are (finite or infinite) sets of variable names.

4.3 Trajectories

Trajectories are total functions that assign system states from the state space to time instants, i.e. elements of $\mathit{Time} \rightarrow \Sigma$, where Σ is the state space. Trajectories are used to describe possible dynamic behaviours of a system. We follow the lines of Duration Calculus [ZHR91], Extended Duration Calculus [ZRH93], and Timed Regular Expressions [ACM96] and restrict attention to trajectories that have a finite number of discontinuities over every finite interval of *Time*. Such trajectories are called *piecewise continuous*. In the special case that the state space Σ is discrete they are also called *finitely variable*. In the remainder, trajectories are always taken to be piecewise continuous. The set of piecewise continuous trajectories over state space Σ is denoted $\mathit{Traj}(\Sigma)$ or *Traj* whenever Σ is clear from the context.

[2] These are not the reachable states in a technical sense, but a set covering those which is often taken to be the Cartesian product of the state spaces of the 'elementary' components of the system. Frequently, only a small fraction of those states is really reachable during system evolution.

4.4 Observations

A finite observation (or observation for short) is a chunk of a trajectory that covers a finite time interval. Formally, the set of *(finite) observations* associated to a set *Traj* of trajectories is $Obs_{Traj} \stackrel{\text{def}}{=} Traj \times TimeInterval$. We will drop the subscript from Obs_{Traj} whenever the set *Traj* of trajectories is clear from the context.

4.5 Timed regular expressions

A convenient way to describe certain sets of observations is the timed automata model of R. Alur and D. Dill [AD94]. Timed automata are finite automata equipped with clocks, which can be reset upon transitions and can be compared to integer constants in guard expressions in order to impose time-dependent constraints upon the enabledness of the guarded transitions. In analogy to regular languages, which are the languages recognized by finite automata, the languages recognized by timed (finite) automata are called *timed regular languages*.

Classically, timed automata are not interpreted by observations, but by timed traces. A *(finite) timed trace* over an alphabet α is a string $\langle (a_0, t_0), (a_1, t_1), \ldots, (a_n, t_n) \rangle \in (\alpha \times Time)^*$ with $t_0 = 0$ and $\forall i \in \mathbb{N}_{<n} \,.\, t_i < t_{i+1}$.[3] The interpretation is that letter a_i is recognized at time instant t_i. An observation-based interpretation of timed automata can be easily derived from their trace-based standard interpretation by assigning to each timed trace $\langle (a_0, t_0), (a_1, t_1), \ldots, (a_n, t_n) \rangle$ the set

$$O\left(\langle (a_0, t_0), (a_1, t_1), \ldots, (a_n, t_n) \rangle\right) \stackrel{\text{def}}{=} \left\{ (tr, [t_0, t_n]) \;\middle|\; \begin{array}{l} tr \in Traj(\alpha), \\ \forall i \in \mathbb{N}_{<n} \,.\, \forall t \in (t_i, t_{i+1}] \,.\, tr(t) = a_i \end{array} \right\}$$

of left-continuous observations. Lifting this to timed automata by assigning to a timed automaton A the set

$$O(A) \stackrel{\text{def}}{=} \bigcup_{t \text{ is accepted by } A} O(t)$$

we obtain an observation-based interpretation of timed automata such that certain sets of observations can be characterized through corresponding timed automata that recognize them.

However, the observations characterize only the externally visible behaviour of the automata such that the necessity to construct an explicit state set of an automaton in order to describe just an observation set is frequently felt to be a burden. This is in analogy to the untimed regular languages where representation by regular expressions is often found to be much more convenient than construction of recognizing automata. The quest for a similar representation of timed regular languages has been recently settled by E. Asarin, P. Caspi, and O. Maler in [ACM96], where they define a notion of *timed regular expressions* that characterizes the observation sets recognized by timed automata in very much

[3] Some authors prefer non-decreasing instead of strictly increasing time sequences. We follow [ACM96] and [Wil94b] in using strictly increasing sequences, which allows us to build upon their results.

the same way that regular expressions characterize the languages accepted by finite automata. It is worthwhile to note that their construction characterizes the observation sets $O(A)$ recognized by timed automata and *not* the sets of timed traces that timed automata recognize according to the classical interpretation. This makes a difference, as in timed traces the same symbol may occur multiple times in a row, which remains invisible in the associated observations. E.g., $\langle(0,\mathrm{a}),(0.75,\mathrm{a}),(2.1,\mathrm{b})\rangle \neq \langle(0,\mathrm{a}),(2.1,\mathrm{b})\rangle$, but $O(\langle(0,\mathrm{a}),(0.75,\mathrm{a}),(2.1,\mathrm{b})\rangle) = O(\langle(0,\mathrm{a}),(2.1,\mathrm{b})\rangle)$.[4]

The syntax of timed regular expressions resembles that of untimed regular expressions, except that a means to control time progress is added and that — as timed regular languages are in contrast to untimed ones not closed under complementation [AD94] — intersection is explicitly present.

Definition 4.1 (Timed regular expression). *Let α be a finite alphabet. The set of* timed regular expressions *over α is defined by the context-free grammar*

$$\begin{array}{rcl}
\langle \mathit{timed\,reg\,exp}\rangle & ::= & \langle \mathit{letter}\rangle \mid \\
 & & \langle \mathit{timed\,reg\,exp}\rangle \vee \langle \mathit{timed\,reg\,exp}\rangle \mid \\
 & & \langle \mathit{timed\,reg\,exp}\rangle \wedge \langle \mathit{timed\,reg\,exp}\rangle \mid \\
 & & \langle \mathit{timed\,reg\,exp}\rangle \cdot \langle \mathit{timed\,reg\,exp}\rangle \mid \\
 & & \langle \mathit{timed\,reg\,exp}\rangle^{*} \mid \\
 & & \langle \mathit{timed\,reg\,exp}\rangle_{\langle \mathit{timing\ constraint}\rangle} \\
\langle \mathit{letter}\rangle & ::\in & \alpha \\
\langle \mathit{timing\,constraint}\rangle & ::= & [\langle \mathit{bound}\rangle, \langle \mathit{bound}\rangle] \mid \\
 & & (\langle \mathit{bound}\rangle, \langle \mathit{bound}\rangle] \mid \\
 & & [\langle \mathit{bound}\rangle, \langle \mathit{general\,bound}\rangle) \mid \\
 & & (\langle \mathit{bound}\rangle, \langle \mathit{general\,bound}\rangle) \\
\langle \mathit{bound}\rangle & ::\in & \mathbb{N} \\
\langle \mathit{general\,bound}\rangle & ::\in & \mathbb{N} \cup \{\infty\}\ .
\end{array}$$

Timed regular expressions are interpreted by sets of observations. A letter $a \in \alpha$ denotes

[4]However, it should be noted that in the original definition of Asarin, Caspi, and Maler, timed automata and timed regular expressions are interpreted by a slightly different notion of observation which is called 'signal' by the authors. Signals are *partial* functions from *Time* to α that have a left-open, right-closed interval as domain. The set of signals assigned to a timed trace $\langle(a_0,t_0),(a_1,t_1),\dots,(a_n,t_n)\rangle$ in [ACM96] is

$$O_{\mathrm{ACM}}(A) \stackrel{\mathrm{def}}{=} \left\{ tr\big|_{(t_0,t_n]} \;\middle|\; \begin{array}{l} tr \in \mathit{Traj}(\alpha), \\ \forall\, i \in \mathbb{N}_{<n} \,.\, \forall\, t \in (t_i,t_{i+1}] \,.\, tr(t) = a_i \end{array} \right\}\ .$$

But the theory developed in [ACM96] does not depend on this particular shape of the domain of observations. For technical reasons we prefer to use the same model class for timed regular expressions and for Duration Logic (chapter 5) and therefore use observations rather than signals, i.e. O instead of O_{ACM}. For every timed automaton A, the observation set $O(A)$ is derived from the signal set $O_{\mathrm{ACM}}(A)$ by application of the mapping $C : \mathcal{P}(\mathit{Signal}) \to \mathcal{P}(\mathit{Obs})$ defined by

$$C(S) \stackrel{\mathrm{def}}{=} \left\{ (tr,[0,e]) \;\middle|\; \begin{array}{l} tr \in \mathit{Traj}(\alpha), \\ e \in \mathit{Time}, \\ \exists\, s \in S \,.\, \big(\mathrm{dom}\,(s) = (0,e] \wedge tr\big|_{(0,e]} = s\big) \end{array} \right\}\ .$$

an observation of arbitrary length that has value a within the observation interval.[5]

$$[\![a]\!] \quad \stackrel{\text{def}}{=} \quad \left\{ (tr, [0,e]) \;\middle|\; \begin{array}{l} tr \in \mathit{Traj}(\alpha), \\ e \in \mathit{Time}, \\ \forall\, t \in (0,e]\,.\; tr(t) = a \end{array} \right\} \ .$$

$\vee$, $\wedge$, and $\cdot$ denote union, intersection, and concatenation of observation sets.

$$\begin{array}{rcl} [\![\phi \vee \psi]\!] & \stackrel{\text{def}}{=} & [\![\phi]\!] \cup [\![\psi]\!] \ , \\ [\![\phi \wedge \psi]\!] & \stackrel{\text{def}}{=} & [\![\phi]\!] \cap [\![\psi]\!] \ , \\ [\![\phi \cdot \psi]\!] & \stackrel{\text{def}}{=} & \left\{ (tr, [0,e]) \;\middle|\; \exists\, m \in [0,e]\,.\; \left(\begin{array}{rl} (tr, [0,m]) & \in [\![\phi]\!] \wedge \\ (t \mapsto tr(t+m), [0, e-m]) & \in [\![\psi]\!] \end{array} \right) \right\} \ . \end{array}$$

The star-operator denotes the timed version of the Kleene star, i.e. finitely iterated catenation of the enclosed timed regular expression:

$$[\![\phi^*]\!] \quad \stackrel{\text{def}}{=} \quad \bigcup_{i \in \mathbb{N}} [\![\phi^i]\!] \ ,$$

where ϕ^i denotes $\phi \cdot \phi \cdot \ldots \cdot \phi$ with i occurrences of ϕ for $i > 0$ and ε for $i = 0$. ε is just an abbreviation for an arbitrary timed regular expression that corresponds to all point observations, e.g. for the timed regular expression $a_{[0,0]}$ with arbitrary $a \in \alpha$.
Finally, ϕ_I denotes those interpretations of ϕ that have a length that obeys the constraint I:

$$[\![\phi_I]\!] \quad \stackrel{\text{def}}{=} \quad \left\{ (tr, [0,e]) \;\middle|\; \begin{array}{l} (tr, [0,e]) \in [\![\phi]\!], \\ e \in \tilde{I} \end{array} \right\} \ ,$$

where $\tilde{I}$ is that interval in *Time* that is the natural interpretation of the constraint I. E.g., for $I = (n, m]$ it is $\{x \in \mathit{Time} \mid n < x \leq m\}$.

The design goal of timed regular expressions has been to obtain a timed counterpart of Kleene's theorem relating finite automata and regular expressions [Kle56]. This goal has been achieved for Asarin's, Caspi's, and Maler's interpretation of timed automata and timed regular expressions, as theorems 1 and 4 of [ACM96] show. As our interpretation is derived from theirs by applying the same mapping C to both their interpretation of timed automata and their interpretation of timed regular languages, these results carry over to our interpretation. This yields the following two theorems.

[5] Again, this definition slightly deviates from that given in [ACM96]. Asarin, Caspi, and Maler instead use signals, i.e. partial functions from *Time* to α with a left-open, right-closed interval as domain, as models of timed regular expressions, defining

$$[\![a]\!]_{\mathrm{ACM}} \quad \stackrel{\text{def}}{=} \quad \left\{ tr\big|_{(0,e]} \;\middle|\; \begin{array}{l} tr \in \mathit{Traj}(\alpha), \\ e \in \mathit{Time}, \\ \forall\, t \in (0,e]\,.\; tr(t) = a \end{array} \right\} \ .$$

The relation to our definition is that $[\![\phi]\!] = C([\![\phi]\!]_{\mathrm{ACM}})$ for each timed regular expression ϕ such that the same function C maps both their interpretation of timed automata and their interpretation of timed regular expressions to our counterparts.

Theorem 4.2 (Timed regular expression to timed automaton). *Every set of observations that is definable by a timed regular expression can be accepted by a timed automaton.*

Proof. Theorem 1 of [ACM96] states the same property for Asarin's, Caspi's, and Maler's interpretation of timed automata and timed regular expressions. I.e. for each timed regular expression ϕ there is a timed automaton A_ϕ with $[\![\phi]\!]_{\mathrm{ACM}} = O_{\mathrm{ACM}}(A_\phi)$. As $[\![\phi]\!] = C([\![\phi]\!]_{\mathrm{ACM}})$ and $O(A_\phi) = C(O_{\mathrm{ACM}}(A_\phi))$, this implies $[\![\phi]\!] = O(A_\phi)$, i.e. the set of observations defined by the timed regular expression ϕ can be accepted by a timed automaton. ∎

As the construction given in [ACM96] in support of theorem 1 is effective this suggests that timed regular expressions are an adequate means for description of observation sets that can be recognized by timed automata. By decidability of the emptiness problem for languages accepted by timed automata [AD94, Theorem 4.16] this implies furthermore

Corollary 4.3 (Decidability of emptiness of timed regular expressions). *It is decidable whether a timed regular expression describes an empty set of observations.* ∎

The other direction of the correspondence of timed automata and timed regular expressions is slightly more indirect in that it needs a renaming of the accepted language. I.e. describing the language that a timed automaton recognizes by a timed regular expression may require an extended alphabet which is then mapped back to the original one through a non-injective language morphism. It is currently not known whether this alphabet extension is really necessary or just an artefact of the proof method. Asarin, Caspi, and Maler conjecture that it is necessary, but have not found an example by now.

Theorem 4.4 (Timed automaton to timed regular expression). *Every set of observations that is accepted by a timed automaton is a renaming of a set of observations that is definable by a timed regular expression.*

Proof. Theorem 4 of [ACM96] states the same property for Asarin's, Caspi's, and Maler's interpretation of timed automata and timed regular expressions. As our interpretation of timed automata and of timed regular expressions is derived from theirs by consistently applying the mapping C, a similar argument as in the proof of the previous theorem derives theorem 4.4 from [ACM96, Theorem 4]. ∎

The construction supporting theorem 4 is again effective. Thus, there is no need to distinguish between those observation sets that can be recognized by a timed automaton and those that can be defined by (a renaming of) a timed regular expression. Henceforth, we will call such observation sets *regular observation sets.*

5

Duration Logic

In order to provide a link to the successful work of the ProCoS project [B$^+$89, B$^+$93, BHL$^+$96, BFOR93, HHF$^+$94, HHMO$^+$96] on provably correct development of embedded safety-critical systems, we base our investigation on the same specification formalism, which is dense-time Duration Calculus. Dense-time Duration Calculus, as defined in [ZHR91], is a logic for reasoning about requirements for real-time systems. It is a dense-time variant of Interval Temporal Logic [Mos85, HMM83] that is tailored towards reasoning about durational constraints on time-dependent Boolean-valued states. By avoiding explicit mention of time and of rise and fall times of Boolean-valued states, it provides a high level of abstraction from operational detail when reasoning about requirements for embedded real-time control systems. Case studies carried out within the ProCoS project, e.g. the gas burner system [RRH93], suggest that this level of abstraction can help in narrowing the gap between safety analysis of the overall system and functional specification of the embedded control system. Unfortunately, the abstraction from operational detail is bought at the price of an — except for quite simple subsets — undecidable logic [ZHS93]. We will expand on decidability issues in the next chapter, whereas this part is devoted to an introduction to the syntax and semantics of a particular Duration Calculus. In order to distinguish it from the variety of other Duration Calculi that have been defined in the literature, we will call this subset *Duration Logic* or *DL* for short.

5.1 Syntax

Since its introduction in [ZHR91], many variants of Duration Calculus have been defined [Zho93], some aiming at even increased expressiveness [ZRH93, Ska94, ZL94, ZH96, Pan96], others at investigation of mechanizability aspects of durational calculi and therefore restricting the vocabulary [ZHS93, SS94, Han94b, ZZYL94]. Aiming at a mechanizable design calculus, we follow the second line and present a slight subset of the Duration Calculus defined in [ZHR91]. However, the particular restrictions of our logic are different from those taken in [ZHS93, SS94, Han94b] as well as those of [ZZYL94]. In contrast to them, our subset still allows full treatment of the gas burner case study [RRH93], the primary case study of the ProCoS project. This indicates that our subset offers an interesting vocabulary for specifying embedded real-time controllers. The discussion at the end of this chapter comments on the particular restrictions that have been imposed compared to the original Duration Calculus of [ZHR91].

5.1.1 State variables

The purpose of Duration Logic is to provide a means of reasoning about time-dependent Boolean states. Syntactically, time-dependent Boolean states are represented by variable names from an infinite set *Varname* of identifiers:

$$\langle \mathit{state\,variable} \rangle \quad ::\in \quad \mathit{Varname}$$

5.1.2 State assertions

State assertions express *point properties*, i.e. properties that can be said to be true or false of a time point. The atomic state assertions are just the state variables, and those can be combined by Boolean connectives to form assertions describing more interesting point properties:

$$\begin{array}{rcl} \langle \mathit{state\,assertion} \rangle & ::= & \langle \mathit{state\,variable} \rangle \mid \\ & & \neg \langle \mathit{state\,assertion} \rangle \mid \\ & & (\langle \mathit{state\,assertion} \rangle \vee \langle \mathit{state\,assertion} \rangle) \end{array}$$

Besides the available Boolean connectives $\neg$ and $\vee$ we frequently take the liberty to use the operators $\wedge$, $\Rightarrow$, $\iff$ and the constants `true` and `false`, which designate the usual abbreviations

$$\begin{aligned} (P \wedge Q) &\stackrel{\text{def}}{=} \neg(\neg P \vee \neg Q) \ , \\ (P \Rightarrow Q) &\stackrel{\text{def}}{=} (\neg P \vee Q) \ , \\ (P \iff Q) &\stackrel{\text{def}}{=} ((P \Rightarrow Q) \wedge (Q \Rightarrow P)) \ , \\ \mathtt{true} &\stackrel{\text{def}}{=} (P \vee \neg P) \ , \\ \mathtt{false} &\stackrel{\text{def}}{=} \neg \mathtt{true} \ . \end{aligned}$$

While above syntax uses full parenthization to avoid ambiguities, we will apply the following binding rules to avoid excessive use of parentheses: $\neg$ binds tightest, followed by $\wedge$, $\vee$, $\Rightarrow$, and $\iff$, in that order. Within sequences of the same binary operator, $\Rightarrow$ and $\iff$ bind to the right, while the other binary connectives are anyhow associative and thus need no extra conventions concerning association.

5.1.3 Durations

Durations are expressions that are evaluated over intervals of time, called observation intervals. They yield the accumulated duration that the enclosed state assertion has been true within the observation interval.

$$\langle \mathit{duration} \rangle \quad ::= \quad \textstyle\int \langle \mathit{state\,assertion} \rangle$$

The concrete syntax $\int P$ for the duration of a state assertion P in the observation interval indeed stems from the mechanism used in the semantics to define the accumulated duration:

the accumulated duration is the Riemann integral of P over time along the observation interval, where the truth values `true` and `false` are interpreted as 1 and 0, respectively.[1]

A frequently occurring duration is $\int$`true`, the duration of the tautologous state assertion `true` over the observation interval, i.e. the length of the observation interval. $l \stackrel{\text{def}}{=} \int\mathtt{true}$ is a convenient shorthand.

5.1.4 Atomic duration formulae

The simplest assertions that can be made of durations are that they obey certain relations to constants:[2]

$$\begin{array}{rcl} \langle atomic\,formula\rangle & ::= & \int\langle state\,assertion\rangle < \langle constant\rangle \mid \\ & & \int\langle state\,assertion\rangle > \langle constant\rangle \\ \langle constant\rangle & ::\in & \mathbb{N} \end{array}$$

The meaning of a duration formula $\int P < k$ ($\int P > k$, respectively) is that it holds of an observation interval iff the accumulated duration of state assertion P being true within this observation interval is less than k (exceeds k, respectively).

5.1.5 Duration formulae

Duration formulae are the top-level formulae of Duration Logic. I.e., technically speaking, $\langle formula\rangle$ is the axiom of the context-free grammar of DL. Duration formulae are built from atomic duration formulae using Boolean connectives and the temporal connective '$;$' (pronounced 'chop').

$$\begin{array}{rcl} \langle formula\rangle & ::= & \langle atomic\,formula\rangle \mid \\ & & \neg\langle formula\rangle \mid \\ & & (\langle formula\rangle \vee \langle formula\rangle) \mid \\ & & (\langle formula\rangle ; \langle formula\rangle) \end{array}$$

While the meaning of the Boolean connectives should be obvious, the chop modality, which is inherited from Interval Temporal Logic [Mos85], may need some explanation. A formula $\phi\,;\,\psi$ is true of an observation interval iff the observation interval can be split into a left and a right subinterval s.t. ϕ holds of the left part and ψ of the right part.

Again, we define some useful abbreviations to be used in formulae. First, the set of relational operators in atomic duration formulae is completed, and the set of Boolean connectives and constants is extended.

$$\begin{array}{rcl} \int P \leq k & \stackrel{\text{def}}{=} & \neg\int P > k \ , \\ \int P \geq k & \stackrel{\text{def}}{=} & \neg\int P < k \ , \end{array}$$

[1] Clearly, taking the Riemann integral requires the interpretation of state assertions to be Riemann-integrable over time. We will come to this issue when dealing with the semantics of durations in section 5.2.

[2] We restrict ourselves to non-negative integer constants. It is not quite clear from [ZHR91] whether the original intention was to use non-negative integer or rational constants. However, this makes no essential difference as any formula contains only a finite number of constants s.t. scaling of the time axis can be applied to convert rational to integer constants.

$$
\begin{aligned}
\textstyle\int P = k &\stackrel{\text{def}}{=} \textstyle\int P \leq k \wedge \int P \geq k \ , \\
\textstyle\int P \neq k &\stackrel{\text{def}}{=} \neg \textstyle\int P = k \ , \\
\textstyle\int P = l &\stackrel{\text{def}}{=} \textstyle\int \neg P = 0 \ , \\
(\phi \wedge \psi) &\stackrel{\text{def}}{=} \neg(\neg\phi \vee \neg\psi) \ , \\
(\phi \Rightarrow \psi) &\stackrel{\text{def}}{=} (\neg\phi \vee \psi) \ , \\
(\phi \iff \psi) &\stackrel{\text{def}}{=} ((\phi \Rightarrow \psi) \wedge (\psi \Rightarrow \phi)) \ , \\
\mathtt{true} &\stackrel{\text{def}}{=} (\phi \vee \neg\phi) \ , \\
\mathtt{false} &\stackrel{\text{def}}{=} \neg\mathtt{true} \ .
\end{aligned}
$$

Then, the temporal operators $\Diamond$ and $\Box$, meaning 'in some subinterval of the observation interval' and 'in each subinterval of the observation interval', are defined.

$$
\begin{aligned}
\Diamond\phi &\stackrel{\text{def}}{=} (\mathtt{true}\,;\,\phi\,;\,\mathtt{true}) \ , \\
\Box\phi &\stackrel{\text{def}}{=} \neg\Diamond\neg\phi \ .
\end{aligned}
$$

In the definition of $\Diamond\phi$ we omitted the inner parentheses as chop is associative. As the formula $(\mathtt{true}\,;\,\phi\,;\,\mathtt{true})$ is satisfied on an observation interval iff the observation interval can be split into three adjacent subintervals, the leftmost and the rightmost satisfying $\mathtt{true}$, the middle one satisfying ϕ, $\Diamond\phi$ indeed means 'in some subinterval of the observation interval ϕ holds'. $\Box\phi$ is simply its dual, meaning 'in no subinterval of the observation interval $\neg\phi$', or equivalently 'in each subinterval of the observation interval ϕ'.

Furthermore, we introduce abbreviation $\lceil\,\rceil$, denoting that the observation interval is a point interval, i.e. has length 0, abbreviation $\lceil P \rceil$, denoting that state assertion P holds almost everywhere in the (non-point) observation interval, and ϕ^k, denoting that ϕ holds and the observation interval has length k.

$$
\begin{aligned}
\lceil\,\rceil &\stackrel{\text{def}}{=} l = 0 \ , \\
\lceil P \rceil &\stackrel{\text{def}}{=} \textstyle\int P = l \ \wedge \ l > 0 \ , \\
\phi^k &\stackrel{\text{def}}{=} \phi \ \wedge \ l = k \ .
\end{aligned}
$$

Finally, a leads-to operator $\phi \leadsto \lceil P \rceil$ is defined that denotes that whenever duration formula ϕ is true in a subinterval of the observation interval, then this subinterval is followed by an interval of positive length where state assertion P holds almost everywhere:

$$
\phi \leadsto \lceil P \rceil \stackrel{\text{def}}{=} \neg\Diamond(\phi\,;\,\lceil \neg P \rceil) \ .
$$

In the remainder, the following binding rules are used to avoid excessive use of parentheses: $\neg$ binds tightest, followed by $\Box$, $\Diamond$, $;$, $\wedge$, $\vee$, $\Rightarrow$, and $\iff$, in that order. Within sequences of the same binary operator, $\Rightarrow$ and $\iff$ bind to the right, while the other binary connectives are anyhow associative.

Note that even with the relaxed parenthization rules, the same concrete syntax for Boolean connectives can be used in both state assertions and duration formulae without risking ambiguity, as state assertions occur only within durations, i.e. within the scope of an integral symbol.

Remark. It is worthwhile to note that the same expressiveness can be achieved with an even more restricted set of atomic duration formulae. The atomic duration formulae of the forms $\int P < k$ and $\int P > k$ can be expressed in terms of $\int P > 0$ and $\int P < 1$ using chop and Boolean connectives. A suitable recursively defined set of expansions is

$$\begin{aligned} \int P < 0 &\mathrel{\hat{=}} \texttt{false} \ , \\ \int P < k+1 &\mathrel{\hat{=}} \int P < 1 \, ; \int P < k \qquad \text{for each } k \in \mathbb{N}_{\geq 1} \ , \\ \int P > 1 &\mathrel{\hat{=}} \neg(\int P < 1) \, ; \int P > 0 \ , \\ \int P > k+1 &\mathrel{\hat{=}} \int P > 1 \, ; \int P > k \qquad \text{for each } k \in \mathbb{N} \ . \end{aligned}$$

We will exploit this for the decidability result of section 6.3. •

5.2 Semantics

Duration Logic is interpreted over trajectories

$$\begin{aligned} tr \in \mathit{Traj} &\stackrel{\text{def}}{=} \{tr : \mathit{Time} \to \Sigma \mid tr \text{ finitely variable}\} \ , \text{ with} \\ \Sigma &\stackrel{\text{def}}{=} \mathit{Varname} \to \mathbb{B} \end{aligned}$$

that provide a finitely variable, time-dependent, Boolean-valued valuation of state variables. These trajectories form the models of DL formulae. Satisfaction of a formula ϕ by a trajectory tr is in turn defined as a limit property over a chain of finite chunks from tr called *finite observations* or simply *observations*, where a finite observation is a pair $(tr, [a, b]) \in \mathit{Traj} \times \mathit{TimeInterval}$ of a trajectory and an observation interval. We will successively define when an observation satisfies ϕ, when a trajectory satisfies ϕ, and when ϕ is valid.

5.2.1 Satisfaction by finite observations

Let $(tr, [a, b])$ be an observation, i.e. a pair of a finitely variable, time-dependent, Boolean-valued valuation of state variables and a finite observation interval.

Semantics of state assertions

As state assertions are point properties, to be evaluated in time instants, semantics of state assertions depends on a state. This yields a mapping $[\![\cdot]\!] : \mathit{StateAssertions} \to \Sigma \to \mathbb{B}$ defined by

$$[\![v]\!](\sigma) \stackrel{\text{def}}{=} \sigma(v) \qquad \text{for each state variable } v \in \mathit{Varname},$$

$$
\begin{aligned}
[\![\neg P]\!](\sigma) &\stackrel{\text{def}}{=} \neg[\![P]\!](\sigma) \ , \\
[\![P \vee Q]\!](\sigma) &\stackrel{\text{def}}{=} [\![P]\!](\sigma) \ \vee \ [\![Q]\!](\sigma) \ .
\end{aligned}
$$

Remark. It is worthwhile to note that for each state assertion P, finite variability of tr is inherited by the mapping $[\![P]\!] \circ tr$, as any state assertion contains only finitely many operators and thus depends only on and varies only with finitely many state variables. ●

Durations

Durations are interpreted over observations, yielding the Riemann integral over the enclosed state assertion along the observation interval, where the truth values `false` and `true` are interpreted as 0 and 1, respectively:

$$
\begin{aligned}
[\![\textstyle\int P]\!](tr,[a,b]) &\stackrel{\text{def}}{=} \int_a^b \chi \circ [\![P]\!] \circ tr(t)\,\mathrm{d}t \ , \\
\chi(x) &\stackrel{\text{def}}{=} \begin{cases} 0 & \text{iff } x = \texttt{false}, \\ 1 & \text{iff } x = \texttt{true}. \end{cases}
\end{aligned}
$$

In order to be well-defined, this definition requires $\chi \circ [\![P]\!] \circ tr$ to be Riemann-integrable over the finite observation interval $[a, b]$. Note that finite variability of $[\![P]\!] \circ tr$ is a sufficient condition for Riemann-integrability over any finite interval such that $[\![\int P]\!]$ is well-defined according to the previous remark.

Duration formulae

A duration formula ϕ is either satisfied on an observation $(tr, [a, b])$, which is denoted by $tr, [a, b] \models \phi$, or not, denoted $tr, [a, b] \not\models \phi$.

An atomic duration formula is satisfied by a finite observation iff the enclosed duration, evaluated over that observation, satisfies the constraint.

$$
\begin{aligned}
tr,[a,b] \models \textstyle\int P < k &\stackrel{\text{def}}{\iff} [\![\textstyle\int P]\!](tr,[a,b]) < k \ , \\
tr,[a,b] \models \textstyle\int P > k &\stackrel{\text{def}}{\iff} [\![\textstyle\int P]\!](tr,[a,b]) > k \ .
\end{aligned}
$$

The interpretation of Boolean connectives is standard.

$$
\begin{aligned}
tr,[a,b] \models \neg\phi &\stackrel{\text{def}}{\iff} tr,[a,b] \not\models \phi \ , \\
tr,[a,b] \models (\phi \vee \psi) &\stackrel{\text{def}}{\iff} tr,[a,b] \models \phi \quad \text{or} \quad tr,[a,b] \models \psi \ .
\end{aligned}
$$

A formula $\phi\,;\psi$ is true of a finite observation iff the underlying observation interval can be split into a left and a right subinterval such that for the resulting finite observations ϕ holds of the left part and ψ of the right part.

$$
tr,[a,b] \models (\phi;\psi) \stackrel{\text{def}}{\iff} \exists\, m \in \mathit{Time}\,.\begin{pmatrix} a \leq m \leq b & \text{and} \\ tr,[a,m] \models \phi & \text{and} \\ tr,[m,b] \models \psi & \end{pmatrix}
$$

Note that the chop modality of Duration Calculus can reference subintervals of the current observation interval only. Together with the notion of validity of a formula introduced in definition 5.4 below, this has the consequence that Duration Calculus cannot express liveness properties. This inability to deal with qualitative progress has sometimes been regretted, although it has been argued that useful progress properties in real-time systems are usually quantitatively bounded, i.e. assign a time-bound to the occurrence of the desired action, and are thus expressible as safety properties in metric-time logics. Consequently, extensions of Duration Calculus that can reference other time intervals besides subintervals of the current observation interval have been developed. In particular, J.-U. Skakkebæk in [Ska94] proposed an extension of Duration Calculus with Venema's outward chop operators [Ven91] and M. R. Hansen examined an extension allowing observation intervals of negative length [Rav95a]. However, restriction to a chop modality that references only subintervals gives rise to some pleasant properties that have prevented the Duration Calculus community from migrating to the extended calculi for gaining the — as it is perceived — minor advantage of dealing with qualitative progress, i.e. liveness. Among those properties are invariance against shifting the absolute position in time and invariance against applying changes to the trajectory:

Lemma 5.1. *Satisfaction of a formula by finite observations is stable against consistently shifting the observation interval and the trajectory along the time line, as well as changing the trajectory outside the observation interval and/or changing the trajectory on sets of time instants of measure 0:*

1. $tr, [a,b] \models \phi$ *iff* $(t \mapsto tr(t+x)), [a-x, b-x] \models \phi$, *provided that* $x \leq a$,
2. $tr, [a,b] \models \phi$ *iff* $tr', [a,b] \models \phi$, *provided that* $tr\big|_{[a,b]} = tr'\big|_{[a,b]}$, *and*
3. $tr, [a,b] \models \phi$ *iff* $tr', [a,b] \models \phi$, *provided that* $tr \approx tr'$.

Proof. By straightforward induction on the syntactic structure of ϕ, proving the slightly stronger propositions that for each subinterval $[a',b'] \subseteq [a,b]$ the equivalences

1. $tr, [a',b'] \models \phi$ iff $(t \mapsto tr(t+x)), [a'-x, b'-x] \models \phi$, provided that $x \leq a$,
2. $tr, [a',b'] \models \phi$ iff $tr', [a',b'] \models \phi$, provided that $tr\big|_{[a',b']} = tr'\big|_{[a',b']}$, and
3. $tr, [a',b'] \models \phi$ iff $tr', [a',b'] \models \phi$, provided that $tr \approx tr'$.

hold. The extension of the propositions to arbitrary subintervals of $[a,b]$ is necessary to deal with chop, which references subintervals. ∎

Note that it is crucial to the proof that chop can reference subintervals of the observation interval only. Indeed, only property 3 is valid for the proposed extensions of Duration Calculus that can reference other time intervals than subintervals. Property 1 could be made true by changing to a model where time is unbounded in both directions, using $Time = \mathbb{R}$, but this comes at the price of burying other pleasant properties.

5.2.2 Satisfaction by infinite trajectories

Ultimately, we are interested in the infinite trajectories that satisfy a formula ϕ. The idea is that a trajectory tr satisfies ϕ iff no finite prefix of tr violates ϕ, i.e. iff each observation from the chain $\{(tr, [0, t]) \mid t \in \mathit{Time}\}$ satisfies ϕ.

Definition 5.2 (Satisfaction). *A duration formula ϕ is* satisfied by a trajectory *tr, denoted $tr \models \phi$, iff $tr, [0, t] \models \phi$ for each $t \in \mathit{Time}$.*

The trajectories satisfying a formula ϕ are also called *the models of ϕ*. The set of models of ϕ is denoted $\mathcal{M}[\![\phi]\!]$.

5.2.3 Satisfiability and validity

Having defined satisfaction of a duration formula both on finite observations and on trajectories, it is natural to derive the corresponding two notions of satisfiability.

Definition 5.3 (Satisfiability). *A duration formula ϕ is called* finitely satisfiable *iff there is an observation $(tr, [a, b])$ that satisfies ϕ. A duration formula ϕ is* satisfiable *iff there is at least one trajectory satisfying ϕ.*

From definition 5.2 and lemma 5.1, the corresponding two notions of validity, i.e. satisfaction by all finite observations and satisfaction by all trajectories, can be easily seen to coincide such that we will not distinguish between finite validity and validity.

Definition 5.4 (Validity). *A duration formula ϕ is called* valid *iff every trajectory satisfies ϕ (or, equivalently, iff every finite observation finitely satisfies ϕ).*

The following relationships between these terms follow immediately from the definitions.

Lemma 5.5. *Let ϕ be a duration formula. Then*

- *satisfiability of ϕ implies finite satisfiability of ϕ,*
- *validity of ϕ implies satisfiability of ϕ,*
- *ϕ is finitely satisfiable if and only if it is satisfiable by an observation of the form $(tr, [0, t])$, and*
- *ϕ is valid if and only if $\neg\phi$ is not finitely satisfiable.*

Proof. Except for the third property, these properties are trivial consequences of the quantifications over models incorporated in the definitions of finite satisfiability, satisfiability, and validity. The third property is a consequence of lemma 5.1. ∎

5.3 Discussion

As previously mentioned, Duration Logic is only a subset of the original Duration Calculus defined in [ZHR91]. Indeed, the only restriction is that durations cannot be combined through arithmetic operations to form duration expressions occurring in atomic duration formulae.[3] In the original calculus, duration expressions may be built from durations and constants using arithmetic operators. Duration expressions are then combined to atomic formulae of the form $\langle duration\,expression\rangle\langle relation\rangle\langle duration\,expression\rangle$. This allows a variety of quantitative relations between durations, like $3\int P = 5\int Q - \int R$, to be expressed, whereas in our subset durations can only be compared to constants. Our subset thus has some similarity to the subsets examined in [ZHS93], which all disallow arithmetic expressions to be built from durations.

In that article, the convention has been introduced to name subsets of Duration Calculus after the forms of atomic duration formulae they contain. Using this convention, DL would be classified as $\{\int P < k, \int P > k\}$, which can be seen to be different from the subsets $\{\lceil P \rceil\}$, $\{\lceil P \rceil, l = k\}$, $\{\int P = \int Q\}$, and $\{\lceil P \rceil, \forall x, l = x\}$[4] studied there. However, as can be seen from the abbreviations in section 5.1.5, formulae of the forms $\lceil P \rceil$ and $l = k$ are encodable in DL such that it covers the subsets $\{\lceil P \rceil\}$ and $\{\lceil P \rceil, l = k\}$. Indeed, it is not hard to see that DL is strictly more expressive than these two subsets, which fail to have a notion of accumulated duration, and is incomparable to $\{\int P = \int Q\}$ and $\{\lceil P \rceil, \forall x, l = x\}$. What distinguishes DL from all the subsets studied in [ZHS93], even the most expressive ones, is that it is expressive enough to cover all constructs used in the gas-burner case study [RRH93, HHF$^+$94], one of the primary case-studies carried out in the ProCoS project.

Given that the gas-burner case study has gained considerable interest in the formal methods community and has become a benchmark for real-time formalism in the meantime, this provides evidence that the subset of Duration Calculus we deal with is of interesting expressiveness. However, this is a pragmatic argument, and the careful reader might like to see whether this argument can be substantiated by showing that other formalisms of useful expressiveness can be encoded in our formalism. We refer the reader to appendix A, where reachability problems of timed transition tables with stop-watches [HKPV95] are encoded in DL.

The reader interested in the practical applicability of Duration Calculi for specification and design of real-time systems is referred to the case studies carried out in connection to the ProCoS project, as documented in [ZHRR92, HZS92, RRH93, BFOR93, HHF$^+$94, PR94]. An extensive study of applying a Duration Calculus to different levels of system design is reported in [Rav95b]. An account of the logical foundations of Duration Calculi, particularly the relationship to Interval Temporal Logic and to modal logics, can be found in [HZ95]. Decidability issues of Duration Calculi have been explored in [ZHS93], and we will provide further results in the next chapter.

[3] The reader familiar with Duration Calculus may also have missed rigid variables and quantification over them, but those were later additions not present in [ZHR91], and are indeed irrelevant to at least the gas-burner case study [RRH93].

[4] This denotes, slightly deviating from the general scheme, a subset featuring real-valued rigid variables with quantification and having atomic formulae of the forms $\lceil P \rceil$ and $l = x$, where x is a rigid variable.

6

Decidability of Duration Logic

Among the good reasons industry has for being reluctant against introduction of formal methods into their development processes, the need for extraordinarily skilled workers and the cost of those particular activities that call for them is a predominant one. One of the most demanding tasks in this respect certainly is formal verification. Therefore, it has become a popular (but not-at-all universally agreed) belief in the formal methods community that availability of "key-press" verification techniques could be a crucial factor for industrial takeover of a certain method. We will not enter into this discussion; instead, we will just investigate the prospects of methods based on Duration Logic for having such techniques.

A practicing engineer may accept a verification method as being *key-press verification* if and only if

1. it is fully automatic, i.e. works without user assistance once the design and the property to be verified have been entered,
2. it is sound, i.e. will not claim a property to be satisfied unless it really is, and
3. it is *sufficiently* complete, i.e. does repel only such a small fraction of the correct designs that development costs are not considerably increased by respecting the advice of the verification tool.

However, property 3 can hardly be formalized. Hence, we will base our investigation of the mechanizability of DL on the slightly stronger condition that

3' the verification method is complete, i.e. succeeds on *any* true property.

Properties 1, 2, and 3' together define the logical concept of *decidability*. Hence, we will have to investigate whether Duration Logic is decidable, or, if not, whether there are restrictions to its model class that could be justified by general properties of the application domain of DL and that suffice to ensure decidability.

Naturally, decidability of dense-time Duration Calculi has been studied before. E.g., Zhou Chaochen, Michael R. Hansen, and Peter Sestoft have shown that validity of formulae from fragment $\{\lceil P \rceil, l = k\}$ of Duration Calculus is undecidable in the standard setting where the models are finitely variable trajectories over dense time [ZHS93, Theorem 11]. As this subset of Duration Calculus is a subset of Duration Logic, the undecidability result carries over to DL. However, it is questionable whether the full range of finitely variable trajectories should be regarded crucial to validity of a DL formula. Any technical system is band-limited. Thus, the possibility to have arbitrarily short transients, which is offered by finitely variable models, is unimportant for describing technical systems. Based on this insight, Thomas Wilke has, for the monadic second order logic of temporal distance, studied n-bounded variability of trajectories [Wil94b]. A trajectory is of variability $n \in \mathbb{N}$ iff it has at most n discontinuities over any unit interval of time. In the case of the monadic second order logic of temporal distance, restriction to models of finitely bounded variability gains decidability, whereas the same logic is undecidable over the full set of finitely variable models. As the proof of theorem 11 of [ZHS93] crucially depends on unbounded variability, the same could be true for the Duration Calculi. Indeed, it is true for subset $\{\lceil P \rceil, l = k\}$ as we will see in section 6.4, but it is not true for DL itself, as will be shown in section 6.2.

We start investigation of decidability aspects by deducing undecidability of DL from the corresponding property of an even more reduced fragment of Duration Calculus that was (among others) studied in [ZHS93]. We proceed in section 6.2 with showing that in contrast to quite some other dense-time logics, decidability on finitely variable and on n-bounded models coincides for Duration Logic for each $n \geq 2$. Hence, exploitation of band-limitedness alone does not suffice to overcome undecidability of DL. Nevertheless, we demonstrate in section 6.3 that decidability of Duration Logic over dense time is not completely hopeless. We show that DL is decidable over dense time if the model class is restricted to time-wise discrete models, where a trajectory is time-wise discrete iff it has discontinuities only in time instants which are multiples of the time unit. Finally, we complete the picture provided by Zhou Chaochen, M. R. Hansen, and P. Sestoft in [ZHS93] by adding the decidability results for n-bounded and for time-wise discrete models for their subsets of Duration Calculus.

6.1 Undecidability on finitely variable models

In [ZHS93], Zhou Chaochen, M. R. Hansen, and P. Sestoft examine various subsets of full Duration Calculus for their decidability over dense and over discrete time. In their terminology, 'dense time' refers to the validity problem for formulae over the set of finitely variable trajectories over non-negative real time and thus refers to the setting we are interested in. Among the subsets they study is the subset $\{\lceil P \rceil, l = k\}$, which denotes the set of formulae built from primitive formulae of the forms $\lceil P \rceil$ and $l = k$ using Boolean connectives and chop. This subset of Duration Calculus can be easily seen to be contained in DL as formulae of the form $\lceil P \rceil$ can be encoded as $\int \mathtt{true} > 0 \wedge \neg \int \neg P > 0$ and as formulae of the form $l = k$ can be encoded as $\neg(\int \mathtt{true} < k \vee \int \mathtt{true} > k)$. Hence, undecidability of the $\{\lceil P \rceil, l = k\}$-fragment of Duration Calculus, as reported in [ZHS93, Theorem 11], implies

Theorem 6.1 (Undecidability of DL over finitely variable models). *Validity of formulae is in general undecidable for DL over finitely variable models.*

Proof. Via the encoding described above, any formula ϕ of the $\{\lceil P \rceil, l = k\}$-fragment can be encoded by a formula $\bar{\phi}$ in fragment $\{\int P < k, \int P > k\}$ such that ϕ is valid iff $\bar{\phi}$ is valid.

Hence, [ZHS93, Theorem 11], which states that subset $\{\lceil P \rceil, l = k\}$ is undecidable, implies undecidability of DL. ∎

It is illustrative to see how the underlying theorem 11 of [ZHS93] is proven. The crucial step is to encode the counter values of a two counter machine by a corresponding number of state changes in intervals of some fixed length k. I.e. a counter value c of counter x is encoded through an observation satisfying

$$l = k \wedge (\lceil \neg x \rceil \,;\, \lceil x \rceil \,;\, \lceil \neg x \rceil \,;\, \lceil x \rceil \,;\, \lceil \neg x \rceil \,;\, \lceil x \rceil \,;\, \ldots \,;\, \lceil \neg x \rceil \,;\, \lceil x \rceil \,;\, \lceil \neg x \rceil)$$

with c occurrences of $\lceil x \rceil$. Incrementing is performed through copying each phase exactly k time units in the future and splitting the last $\lceil \neg x \rceil$-phase into three phases $\lceil \neg x \rceil \,;\, \lceil x \rceil \,;\, \lceil \neg x \rceil$, and vice versa for decrementing. This encoding clearly requires that trajectories may have an unbounded (finite) number of discontinuities over compact time intervals, as the counter value c is unbounded.

An alternative proof of theorem 6.1, which puts indeed less demands on the variability allowed in trajectories, can be found in appendix A, where reachability problems for timed transition tables with stop-watches are encoded in DL. Having done so, undecidability of these reachability problems (cf. e.g. [HKPV95, Theorem 4.1]) implies undecidability of DL, thus proving theorem 6.1.

6.2 Undecidability on models of finitely bounded variability

It is questionable whether the full range of finitely variable trajectories should be regarded crucial to validity of DL formulae. As any technical system is band-limited, the possibility to have arbitrarily short transients, which is offered by finitely variable models, is unimportant for describing technical systems. A more restrictive constraint on variability that reflects the idea of band-limitedness is *n-bounded variability.*

Definition 6.2 (n-bounded variability). *Let $n \in \mathbb{N}$. A trajectory tr has n-bounded variability iff tr has at most n discontinuity points within any unit interval of time. The set of trajectories of n-bounded variability is denoted $Traj_n$.*

A formula ϕ is called valid on n-bounded models *iff each trajectory of n-bounded variability is a model of ϕ.*

Clearly, for any $n \in \mathbb{N}$, the set of n-bounded trajectories is a subset of the finitely variable trajectories, such that validity of a formula on n-bounded trajectories is less demanding than validity on the full set of models. The monadic second order logic of temporal distance is an example of a logic where this makes a difference to decidability: it is undecidable on finitely variable models, but decidable on n-bounded models (for any $n \in \mathbb{N}$) [Wil94b]. However, the same is not true for Duration Logic, as we will show in this section. Note that this is a new result, as the proof of undecidability of the $\{\lceil P \rceil, l = k\}$-fragment of Duration Calculus on finitely variable models in [ZHS93, theorem 11] crucially depends on unbounded variability, and thus does not carry over to the restricted model class of n-bounded trajectories.

Roughly, the argument for undecidability of Duration Logic on n-bounded models put forward in the remainder proceeds as follows: for each formula ϕ of Duration Logic, a formula ϕ_{slow} is constructed such that ϕ_{slow} is valid on n-bounded models (for any $n \geq 2$) if and only if ϕ itself is valid on the full set of finitely variable models. Then, undecidability of validity of DL formulae on the full set of finitely variable trajectories implies undecidability of validity of DL formulae on n-bounded models.

The construction of ϕ_{slow} from a formula ϕ is amazingly simple. Let tf be a state variable that does not occur in ϕ. ϕ_{slow} is then derived from ϕ by replacing any duration $\int P$ occurring in ϕ by the duration $\int(P \wedge tf)$.[1] The intuitive effect of this procedure is that tf plays the role of a flag indicating which parts of the trajectory are relevant to durations (and thus to satisfaction of the formula). Whenever tf is true, the duration $\int P$ actually accumulates. But if tf is false over some interval of time then this interval does not contribute to any duration in ϕ_{slow} such that registration of the passage of time effectively pauses during this period.[2] Hence, introduction of tf allows us to introduce 'pauses' into the trajectories which do not affect satisfaction of the formula ϕ_{slow}, but can serve to reduce variability of the trajectory to a finitely bounded value.

The idea of pauses in the trajectory that do not affect satisfaction can be formalized by a compression operation that removes those pauses. It is a binary relation

$$\mathit{compress} \subseteq \mathit{Obs}_{\mathit{Traj}} \times \mathit{Obs}_{\mathit{Traj}}$$

between finite observations that is defined by

$$(tr, [a,b])\ \mathit{compress}\ (tr', [a',b']) \overset{\mathrm{def}}{\Longleftrightarrow} a' = tv_{tr}(a) \ \wedge\ b' = tv_{tr}(b) \ \wedge\ tr = tr' \circ tv_{tr} \ ,$$

where $tv_{tr}(t) \overset{\mathrm{def}}{=} [\![\int tf]\!](tr, [0,t])$ is the accumulated time that the trajectory tr was not 'pausing' up to time t. Note that tv_{tr} is continuous and monotonic as, according to the semantics of DL, $tv_{tr}(t) = \int_0^t \chi \circ [\![tf]\!] \circ tr(t)\,\mathrm{d}t$ and χ is non-negative.

The effect of *compress* is illustrated in figure 6.1 on the facing page. An interesting fact about compress is that it transfers models of ϕ_{slow} into models of ϕ.

Lemma 6.3. *$tr, [a,b] \models \phi_{\mathrm{slow}}$ iff $tr', [a',b'] \models \phi$ for any observations $(tr, [a,b])$ and $(tr', [a',b'])$ with $(tr, [a,b])$ compress $(tr', [a',b'])$.*

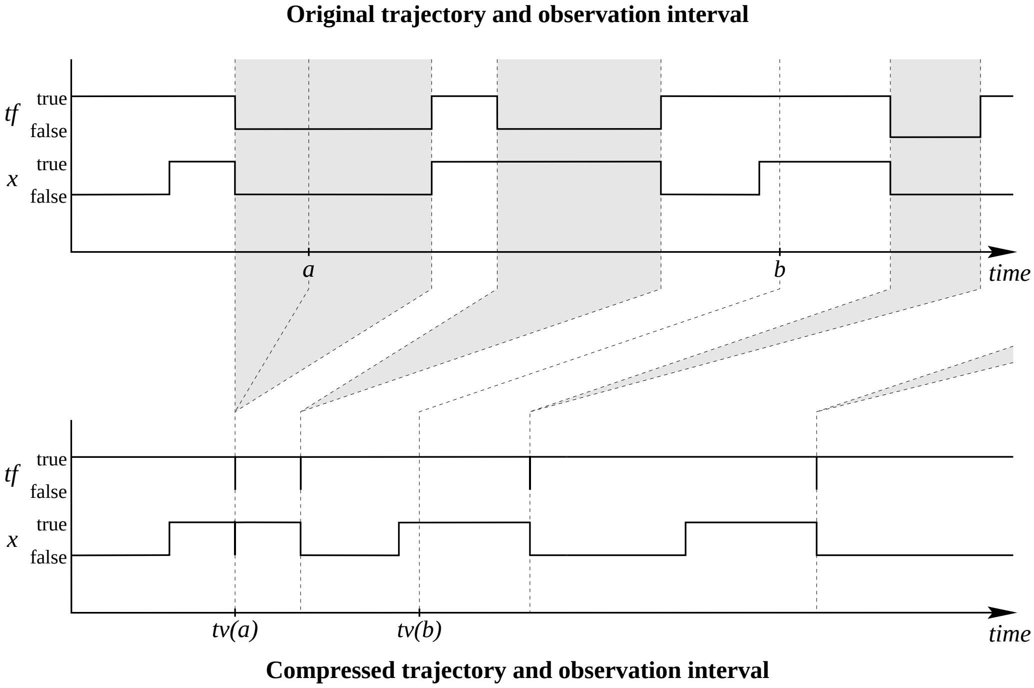
Proof. By induction on the structure of ϕ.

We start with the base cases, which are $\phi = \int P < k$ and $\phi = \int P > k$. Assume that $(tr, [a,b])$ and $(tr', [a',b'])$ are arbitrary observations with $(tr, [a,b])$ *compress* $(tr', [a',b'])$.

If ϕ is of the form $\int P < k$ then ϕ_{slow} is $\int(P \wedge tf) < k$. Due to the semantics of Duration Logic, showing that $tr, [a,b] \models \int(P \wedge tf) < k$ if and only if $tr', [a',b'] \models \int P < k$ requires showing that $[\![\int(P \wedge tf)]\!](tr, [a,b]) < k$ if and only if $[\![\int P]\!](tr', [a',b']) < k$. Hence, it suffices to show that $[\![\int(P \wedge tf)]\!](tr, [a,b]) = [\![\int P]\!](tr', [a',b'])$, which is accomplished by the following calculation.

[1] Note that thereby the duration l is also modified, as l is just an abbreviation for $\int \mathtt{true}$ and thus is replaced by $\int tf$.

[2] Hence the name tf, which is an acronym for *time flow*.



Finite observations are compressed by removing phases where tf is false throughout (the shaded parts) from the trajectory and by adjusting the bounds a and b of the observation interval accordingly. The adjusted bounds are $tv(a)$ and $tv(b)$.

Figure 6.1: Compressing an observation by removal of phases that are irrelevant to satisfaction of ϕ_{slow}

$$
\begin{aligned}
&[\![\textstyle\int(P \wedge tf)]\!](tr,[a,b]) \\
= &\int_a^b (\chi \circ [\![P \wedge tf]\!] \circ tr)\,(t)\,\mathrm{d}t && \text{[Semantics of DL]} \\
= &\int_a^b (\chi \circ [\![P]\!] \circ tr)\,(t) \cdot (\chi \circ [\![tf]\!] \circ tr)\,(t)\,\mathrm{d}t && \text{[Semantics of DL, def. of } \chi\text{]} \\
= &\int_a^b (\chi \circ [\![P]\!] \circ tr)\,(t) \cdot \frac{\mathrm{d}}{\mathrm{d}t} tv_{tr}(t)\,\mathrm{d}t && \text{[Def. of } tv_{tr}\text{]} \\
= &\int_a^b \left(\chi \circ [\![P]\!] \circ tr' \circ tv_{tr}\right)(t) \cdot \frac{\mathrm{d}}{\mathrm{d}t} tv_{tr}(t)\,\mathrm{d}t && \text{[Def. of } \textit{compress}\text{]} \\
= &\int_{tv_{tr}(a)}^{tv_{tr}(b)} \left(\chi \circ [\![P]\!] \circ tr'\right)(u)\,\mathrm{d}u && \text{[Integration by substitution]}
\end{aligned}
$$

$$= \int_{a'}^{b'} (\chi \circ [\![P]\!] \circ tr')\,(u)\,\mathrm{d}u \qquad \text{[Def. of } \mathit{compress}\text{]}$$

$$= [\![\textstyle\int P]\!](tr', [a', b']) \qquad \text{[Semantics of DL]}$$

This shows that $tr, [a, b] \models \int(P \wedge tf) < k$ iff $tr', [a', b'] \models \int P < k$. Furthermore, the same calculation proves that $tr, [a, b] \models \int(P \wedge tf) > k$ iff $tr', [a', b'] \models \int P > k$, which completes the base cases.

For the induction step assume that ϕ_1 and ϕ_2 are such that for each pair of observations $((tr, [a, b]), (tr', [a', b'])) \in \mathit{compress}$ it is true that $tr, [a, b] \models \phi_i$ iff $tr', [a', b'] \models \phi_{i\text{slow}}$.

The first case of the induction step is that ϕ is of the form $\neg\phi_1$. Then

$$
\begin{aligned}
& tr, [a, b] \models \phi_{\text{slow}} & \\
\iff\ & tr, [a, b] \models (\neg\phi_1)_{\text{slow}} & \\
\iff\ & tr, [a, b] \models \neg(\phi_{1\text{slow}}) & \text{[Def. of } \phi_{\text{slow}}\text{]} \\
\iff\ & tr, [a, b] \not\models \phi_{1\text{slow}} & \text{[Semantics of DL]} \\
\iff\ & tr', [a', b'] \not\models \phi_1 & \text{[Assumption on } \phi_1\text{]} \\
\iff\ & tr', [a', b'] \models \neg\phi_1 & \text{[Semantics of DL]} \\
\iff\ & tr', [a', b'] \models \phi &
\end{aligned}
$$

which proves the proposition for $\phi = \neg\phi_1$. A similar derivation proves the case $\phi = \phi_1 \vee \phi_2$.

The last case that remains to be checked is $\phi = \phi_1 \,;\, \phi_2$. Then

$$
\begin{aligned}
& tr, [a, b] \models \phi_{\text{slow}} & \\
\iff\ & tr, [a, b] \models (\phi_1 \,;\, \phi_2)_{\text{slow}} & \\
\iff\ & tr, [a, b] \models \phi_{1\text{slow}} \,;\, \phi_{2\text{slow}} & \text{[Def. of } \phi_{\text{slow}}\text{]} \\
\iff\ & \exists\, m \in \mathit{Time}\,.\, \begin{pmatrix} a \le m \le b & \wedge \\ tr, [a, m] \models \phi_{1\text{slow}} & \wedge \\ tr, [m, b] \models \phi_{2\text{slow}} & \end{pmatrix} & \text{[Semantics of DL]} \\
\iff\ & \exists\, m \in \mathit{Time}\,.\, \begin{pmatrix} a \le m \le b & \wedge \\ tr', [tv_{tr}(a), tv_{tr}(m)] \models \phi_1 & \wedge \\ tr', [tv_{tr}(m), tv_{tr}(b)] \models \phi_2 & \end{pmatrix} & \text{[Assumptions on } \phi_1 \text{ and } \phi_2\text{]} \\
\iff\ & \exists\, m \in \mathit{Time}\,.\, \begin{pmatrix} tv_{tr}(a) \le m \le tv_{tr}(b) & \wedge \\ tr', [tv_{tr}(a), m] \models \phi_1 & \wedge \\ tr', [m, tv_{tr}(b)] \models \phi_2 & \end{pmatrix} & \text{[Monotonicity and continuity of } tv_{tr}\text{]} \\
\iff\ & tr', [tv_{tr}(a), tv_{tr}(b)] \models \phi_1 \,;\, \phi_2 & \text{[Semantics of DL]}
\end{aligned}
$$

$$\iff \quad tr', [a', b'] \models \phi_1 \,;\, \phi_2 \qquad \text{[Def. of } \mathit{compress}\text{]}$$

$$\iff \quad tr', [a', b'] \models \phi$$

∎

Having a criterion on how compression of observations transfers satisfaction between ϕ_{slow} and ϕ, it is interesting to examine the inverse operation: 'pumping' of observations in order to reduce their rate of variation. Indeed, pumping can reduce the variability of a given finitely variable observation to any finite degree strictly greater than 1, as the following lemma shows.

Lemma 6.4. *Let tr' be a trajectory with $tr'(t)(tf) = \texttt{true}$ for every $t \in \mathit{Time}$, and let $[a', b'] \in \mathit{TimeInterval}$. Then there is for each $n \in \mathbb{N}_{\geq 2}$ an observation $(tr, [a, b]) \in \mathit{Obs}_{\mathit{Traj}_n}$, i.e. an observation of n-bounded variability, with $(tr, [a, b])\ \mathit{compress}\ (tr', [a', b'])$.*

Proof. As the n-bounded observations are, for $n \geq 2$, a superset of the 2-bounded observations, it suffices to show the conjecture for $n = 2$.

Let $(tr', [a, b]) \in \mathit{Obs}_{\mathit{Traj}}$ with $tr'(t)(tf) = \texttt{true}$ for every $t \in \mathit{Time}$. A 2-bounded observation $(tr, [a, b])$ is easily constructed from $(tr', [a, b])$ by introducing a 'pause' of one time-unit length after each discontinuity in tr': let $D \subset \mathit{Time}$ be the set of discontinuity points of tr'. Now define for each $t \in \mathit{Time}$ and each $v \in \mathit{Varname}$

$$tr(t)(v) \stackrel{\text{def}}{=} \begin{cases} tr'(t')(v) & \text{if } t' + |\{d \in D \mid d < t'\}| = t, \\ \texttt{false} & \text{if no such } t' \in \mathit{Time} \text{ exists.} \end{cases}$$

tr introduces a pause of one time-unit length, where all state variables evaluate to $\texttt{false}$, after each discontinuity in tr'. Hence, its variability is 2-bounded. Furthermore, it is easy to see that $(tr, [a' + |\{d \in D \mid d < a'\}|\,, b' + |\{d \in D \mid d < b'\}|])\ \mathit{compress}\ (tr', [a', b'])$. Note that $a' + |\{d \in D \mid d < a'\}|$ and $b' + |\{d \in D \mid d < b'\}|$ are always finite, as finite variability of tr' implies that $|\{d \in D \mid d < t\}|$ is finite for any $t \in \mathit{Time}$. ∎

Together with lemma 6.3 this implies that any finite model of ϕ has a corresponding n-bounded finite model of ϕ_{slow}, such that finite satisfiability transfers in the following way.

Lemma 6.5. *Let $n \in \mathbb{N}_{\geq 2}$. Then ϕ is finitely satisfiable iff ϕ_{slow} is finitely satisfiable by an n-bounded model.*

Proof. First assume that ϕ_{slow} is satisfied by an n-bounded finite model $(tr, [a, b])$. As all state assertions in ϕ_{slow} are of the form $P \wedge tf$, we may without loss of generality assume that $tr(t)(tf) = \texttt{false}$ implies $tr(t)(v) = \texttt{false}$ for each $t \in \mathit{Time}$ and each $v \in \mathit{Varname}$. Then $(tr, [a, b]) \in \operatorname{dom}(\mathit{compress})$ and thus, according to lemma 6.3, there is a finite model $(tr', [a', b'])$ of ϕ.

Now assume that ϕ is finitely satisfied by $(tr', [a', b'])$. W.l.o.g., $tr'(t)(tf) = \texttt{true}$ for all $t \in \mathit{Time}$, as tf does not occur in ϕ. But then there is an n-bounded observation $(tr, [a, b])$ with $(tr, [a, b])\ \mathit{compress}\ (tr', [a', b'])$ due to lemma 6.4, and $(tr, [a, b])$ satisfies ϕ_{slow} due to lemma 6.3. ∎

As finite satisfiability is dual to validity, an immediate consequence is that validity of a formula ϕ and validity of ϕ_{slow} on n-bounded models coincide.

Corollary 6.6. *Let $n \in \mathbb{N}_{\geq 2}$. ϕ is valid iff ϕ_{slow} is valid on n-bounded models.*

Proof. This fact follows immediately from lemma 6.5, applied to $\neg\phi$, as validity is dual to finite satisfiability (lemma 5.5) and validity on n-bounded models is dual to finite n-bounded satisfiability (analogous). ▟

As validity is undecidable for DL, this in turn implies the desired undecidability result for DL on n-bounded models:

Theorem 6.7 (Undecidability of DL over n-bounded models for $n \geq 2$). *For each natural number $n \geq 2$, validity of DL formulae on n-bounded models is in general undecidable.*

Proof. If validity of DL formulae on n-bounded models were decidable for some $n \geq 2$, then validity of ϕ_{slow} on n-bounded models would be decidable for each DL formula ϕ. According to corollary 6.6 such a procedure would also decide validity of ϕ on the full set of finitely variable models, in contrast to undecidability of DL. ▟

This result can be further generalized to the case $n = 1$ by encoding reachability problems of timed automata with stopwatches in DL, as is done in appendix A. However, this generalization is of only marginal practical interest, as systems featuring trajectories of only 1-bounded variability are hardly ever encountered in practice. The reason is that even though we might be able to use trajectories of 1-bounded variability for modelling sequential systems, parallel composition adds up the variability of causally independent components, such that the class of systems with models of 1-bounded variability is not closed under parallel composition. More generally, any decidability or undecidability result covering only some fixed degree of bounded variability is of very limited practical importance in the field of concurrency. Therefore, and due to the fundamentally different techniques required, we will delay analysis of decidability on 1-bounded models to appendix A, theorem A.7.

Theorem 6.7 above shows that exploitation of band-limitedness, at least as formalized by bounded variability, is insufficient for gaining decidability of full Duration Logic. This is quite unfortunate, as band-limitedness is a general property of reactive systems that holds regardless of the particular implementation technology used. As this common property of all reactive systems fails to yield decidability of DL, we have two options remaining: either we investigate even more restricted model classes which are justifiable by the dynamics of a particular implementation technology, or we restrict attention to subsets of DL that are decidable on n-bounded models.

Both alternatives have their particular drawbacks: while the former restricts the available implementation technology, the latter removes expressiveness from the specification language. Therefore, we will pursue both approaches: the next section deals with decidability on a more restricted model class, whereas section 6.4 analyses syntactic variants of Duration Logic.

6.3 Decidability on time-wise discrete models

We may try to reduce the model class that is regarded crucial to validity of a formula still further. If the devices to be designed are fully synchronous and evenly clocked (which they are not in our context, but we will find means to control the impact of this over-idealization) it makes sense to consider only trajectories that are changing at evenly spaced time instants. This is captured by studying validity of DL formulae over *time-wise discrete trajectories*, which are defined by

Definition 6.8 (Time-wise discreteness). *A trajectory $tr \in Traj$ is called* time-wise discrete *iff it has discontinuities only in time instants which are multiples of the time unit. I.e., tr is time-wise discrete iff*

$$\forall k \in \mathbb{N} \,.\, \forall \delta, \delta' \in (0,1) \,.\, tr(k+\delta) = tr(k+\delta') \quad .$$

The set of time-wise discrete trajectories is denoted $DTraj$.

A formula ϕ is called valid on time-wise discrete models *iff each time-wise discrete trajectory satisfies ϕ. ϕ is said to be* time-wise discretely prefix-satisfiable *iff there is a time-wise discrete trajectory tr and $e \in Time$ such that $tr, [0,e] \models \phi$.*

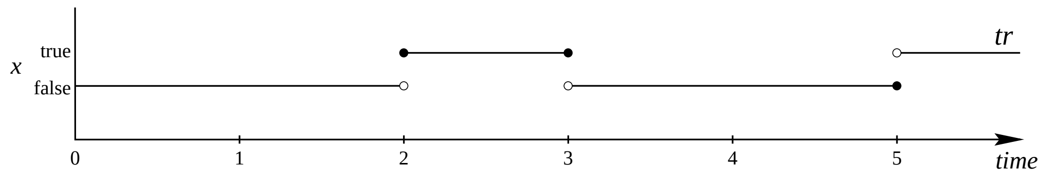
Obviously, validity on time-wise discrete models is a necessary, but not sufficient, condition for validity on the full set of finitely variable models, as the time-wise discrete trajectories form a proper subset of the finitely variable trajectories. Furthermore, in analogy to lemma 5.5, validity and finite satisfiability over time-wise discrete models is again closely related:

Lemma 6.9. *Let ϕ be a duration formula. ϕ is valid over time-wise discrete models if and only if $\neg\phi$ is not time-wise discretely prefix-satisfiable.*

Proof. By definition, a (time-wise discrete) trajectory tr satisfies ϕ iff $tr, [0,e] \models \phi$ for each $e \in Time$. As $tr, [0,e] \models \neg\phi$ iff $tr, [0,e] \not\models \phi$, $\neg\phi$ is time-wise discretely prefix-satisfiable iff there is a time-wise discrete trajectory not satisfying ϕ, i.e. iff ϕ is not valid over time-wise discrete models. ∎

The restriction to time-wise discrete trajectories leads to decidability as we will show in this section. As far as we know, this is the first positive decidability result for a logic featuring a notion of accumulated duration, dense-time chop, and unrestricted negation. The proof proceeds as follows.

1. A *finite* subset $V \subset Varname$ of the state variable names is chosen arbitrarily,
2. for each formula ϕ with $free(\phi) \subseteq V$ it is shown that the set of time-wise discrete observations that finitely satisfy ϕ corresponds to an effectively constructible regular language over the alphabet $V \to \mathbb{B}$,
3. using decidability of the emptiness problem for regular languages this proves decidability of time-wise discrete prefix-satisfiability,



wrd_V maps time-wise discrete trajectories to infinite strings. Given a time-wise discrete trajectory tr it yields the sequence of valuations of variables from V that tr assigns to the integer-aligned unit intervals of time. Hence, given that $V = \{x\}$ and tr is the trajectory depicted above,

$$wrd_V(tr) \quad = \quad \langle x = \mathtt{false}, x = \mathtt{false}, x = \mathtt{true}, x = \mathtt{false}, x = \mathtt{false}, x = \mathtt{true}, \ldots \rangle \ .$$

Note that wrd_V is independent of the border values of the unit intervals and takes the values inside the intervals instead.

Figure 6.2: Mapping a time-wise discrete trajectory to an infinite string

4. by lemma 6.9 this shows decidability of validity over time-wise discrete models.

We start by arbitrarily choosing a finite set $V \subset \mathit{Varname}$. As V is finite, the alphabet $\alpha \stackrel{\text{def}}{=} V \to \mathbb{B}$ is finite. A digitization $wrd_V : \mathit{DTraj} \to \alpha^\omega$ that maps time-wise discrete observations to infinite strings over α is defined by

$$wrd_V(tr)(m) \stackrel{\text{def}}{=} \lim_{t \geq m} tr(t)\big|_V \qquad \text{for each } m \in \mathbb{N}.$$

wrd_V maps a time-wise discrete trajectory tr to the sequence of values which are assigned to the integer-aligned unit-intervals of time by tr. This is illustrated in figure 6.2. Note that wrd_V would be well-defined even for finitely variable trajectories, not only for time-wise discrete ones. However, on time-wise discrete trajectories wrd_V is 'almost bijective' in the following sense: let $tr, tr' \in \mathit{DTraj}$. Then $wrd_V(tr) = wrd_V(tr')$ iff $(t \mapsto tr(t)\big|_V) \approx (t \mapsto tr'(t)\big|_V)$, i.e. wrd_V is injective up to almost-equality on V. Surjectivity of wrd_V is obvious. A consequence is that time-wise discrete trajectories which share the same image under wrd_V satisfy the same DL formulae, provided that the free variables of the formulae are in V:

Lemma 6.10. *Let $tr, tr' \in \mathit{DTraj}$ with $wrd_V(tr) = wrd_V(tr')$ and let $[a, b] \in \mathit{TimeInterval}$. Then*

- *$tr \models \phi$ iff $tr' \models \phi$ and*
- *$tr, [a, b] \models \phi$ iff $tr', [a, b] \models \phi$*

for each formula ϕ with $\mathit{free}(\phi) \subseteq V$.

Proof. Lemma 5.1, property 3. ∎

wrd_V is used for assigning to each DL formula ϕ with $free(\phi) \subseteq V$ a family of languages over α that will turn out to have an effective construction.

Definition 6.11. *Let ϕ be a formula with $free(\phi) \subseteq V$. For each $l, r \in [0, 1)$,*

$$\mathcal{L}_{l,r}(\phi) \stackrel{\text{def}}{=} \left\{ wrd_V(tr)[0, \lfloor e \rfloor] \;\middle|\; \begin{array}{l} tr \in DTraj,\ e \in Time, \\ e \geq l \ \wedge \ e = \lfloor e \rfloor + r, \\ tr, [l, e] \models \phi \end{array} \right\}$$

is the set of digitized finite observations that satisfy ϕ, start in time instant l and end with offset r to an integer time instant.

Through its definition, emptiness of $\mathcal{L}_{l,r}(\phi)$ is related to finite satisfiability of ϕ by observations that obey certain restrictions on their start and end times, namely that they start in l and end with offset r to an integer time instant. The rationale behind this construction is that emptiness of each of the languages $\mathcal{L}_{0,r}(\neg\phi)$ characterizes validity of ϕ over time-wise discrete models:

Lemma 6.12. *Let ϕ be a formula with $free(\phi) \subseteq V$. ϕ is valid over timed-wise discrete models iff $\bigcup_{r \in [0,1)} \mathcal{L}_{0,r}(\neg\phi) = \emptyset$.*

Proof. We start with showing that $\bigcup_{r \in [0,1)} \mathcal{L}_{0,r}(\neg\phi) \neq \emptyset$ iff $\neg\phi$ is time-wise discretely prefix-satisfiable. Therefore assume that $tr, [0, e] \models \neg\phi$ and tr is time-wise discrete. Then, by definition 6.11, $wrd_V(tr)[0, \lfloor e \rfloor] \in \mathcal{L}_{0,e-\lfloor e \rfloor}(\neg\phi)$ and thus $\bigcup_{r \in [0,1)} \mathcal{L}_{0,r}(\neg\phi) \neq \emptyset$. Vice versa, assume that $tr, [0, e] \not\models \neg\phi$ for any $tr \in DTraj$ and $e \in Time$. Then, again by definition 6.11, $\mathcal{L}_{0,r}(\neg\phi) = \emptyset$ for any $r \in [0, 1)$. Hence, $\bigcup_{r \in [0,1)} \mathcal{L}_{0,r}(\neg\phi) \neq \emptyset$ iff $\neg\phi$ is time-wise discretely prefix-satisfiable.

According to lemma 6.9, ϕ is valid over time-wise discrete models iff $\neg\phi$ is not time-wise discretely prefix-satisfiable. Thus, $\bigcup_{r \in [0,1)} \mathcal{L}_{0,r}(\neg\phi) = \emptyset$ iff ϕ is valid over time-wise discrete models. ∎

In the sequel we will reveal a procedure for deciding whether $\bigcup_{r \in [0,1)} \mathcal{L}_{0,r}(\neg\phi) = \emptyset$ such that lemma 6.12 provides a handle to deciding validity of formulae over time-wise discrete models. The revelation proceeds in two steps: We start with a sequence of lemmata that characterize the dependency of $\mathcal{L}_{l,r}(\phi)$ on l, r, and the syntactic structure of ϕ. Then we deduce from that characterization that $\bigcup_{r \in [0,1)} \mathcal{L}_{0,r}(\neg\phi)$ is an effectively constructible regular language for each formula ϕ.

The dependency of $\mathcal{L}_{l,r}(\phi)$ on l, r, and the syntactic structure of ϕ is subject of lemmata 6.13 to 6.18.

Lemma 6.13. *Let P be a state assertion with $free(P) \subseteq V$. Then*

$$\mathcal{L}_{l,r}(\textstyle\int P < 1) = \begin{cases} \widetilde{\neg P}^* \cdot \alpha & \text{if } l = 0 \ , \\ (\widetilde{\neg P}^* \cdot \alpha) \cup (\alpha \cdot \widetilde{\neg P}^*) & \text{if } 0 < l \leq r \ , \\ \alpha \cdot \widetilde{\neg P}^* \cdot \alpha & \text{if } l > r \ , \end{cases}$$

where $\widetilde{P} \stackrel{\text{def}}{=} \left\{ \sigma\big|_V \;\middle|\; \begin{array}{l} \sigma \in \Sigma, \\ [\![P]\!](\sigma) = \texttt{true} \end{array} \right\}$ *is the set of valuations in the alphabet* α *that satisfy the state assertion* P*. For convenience, we define* $\emptyset^* \stackrel{\text{def}}{=} \{\varepsilon\}$*, i.e.* $\widetilde{\texttt{false}}^* = \{\varepsilon\}$*.*

Proof. Figure 6.3 on the facing page illustrates which observations $(tr, [l, e])$ with $l \leq e = \lfloor e \rfloor + r$ satisfy $\int P < 1$ depending on the choice of $l, r \in [0, 1)$. It is straightforward to construct the image of these model classes under the mapping $(tr, [l, e]) \mapsto wrd_V(tr)[0, \lfloor e \rfloor]$. For $l = 0$ this image is $\widetilde{\neg P}^* \cdot \alpha$ such that $\mathcal{L}_{0,r}(\int P < 1) = \widetilde{\neg P}^* \cdot \alpha$. Similarly, for $0 < l \leq r$ it is $(\widetilde{\neg P}^* \cdot \alpha) \cup (\alpha \cdot \widetilde{\neg P}^*)$, yielding the proposition for this case. Finally, for $l > r$ the side condition $l \leq e = \lfloor e \rfloor + r$ implies $e \geq 1$ such that the image of $(tr, [l, e]) \mapsto wrd_V(tr)[0, \lfloor e \rfloor]$ contains only words of length greater or equal to 2 in that case. Together with the structure of the models explained in figure 6.3 on the next page this shows that $\mathcal{L}_{l,r}(\int P < 1) = \alpha \cdot \widetilde{\neg P}^* \cdot \alpha$ whenever $l > r$. ∎

Lemma 6.14. *Let* P *be a state assertion with* $\mathit{free}(P) \subseteq V$*. Then*

$$\mathcal{L}_{l,r}(\textstyle\int P > 0) \quad = \quad \begin{cases} \alpha^* \cdot \widetilde{P} \cdot \alpha^+ & \text{if } \; r = 0 \;, \\ \alpha^* \cdot \widetilde{P} \cdot \alpha^* & \text{if } \; r > l \;, \\ \alpha^{\geq 2} \cap \left(\alpha^* \cdot \widetilde{P} \cdot \alpha^*\right) & \text{if } \; 0 < r \leq l \;, \end{cases}$$

where $\alpha^{\geq 2} \stackrel{\text{def}}{=} \{w \in \alpha^* \mid \mathrm{len}(w) \geq 2\}$*.*

Proof. By a similar inspection of the model class as in the proof of lemma 6.13. ∎

Lemma 6.15. *Suppose* ϕ *is a formula with* $\mathit{free}(\phi) \subseteq V$*. Then*

$$\mathcal{L}_{l,r}(\neg\phi) \quad = \quad \begin{cases} \alpha^+ \setminus \mathcal{L}_{l,r}(\phi) & \text{if } \; l \leq r, \\ \alpha^{\geq 2} \setminus \mathcal{L}_{l,r}(\phi) & \text{if } \; l > r. \end{cases}$$

Proof. For the case $l \leq r$ we show $\mathcal{L}_{l,r}(\neg\phi) = \alpha^+ \setminus \mathcal{L}_{l,r}(\phi)$ by proving that

1. $\mathcal{L}_{l,r}(\phi) \cup \mathcal{L}_{l,r}(\neg\phi) = \alpha^+$ and
2. $\mathcal{L}_{l,r}(\phi) \cap \mathcal{L}_{l,r}(\neg\phi) = \emptyset$.

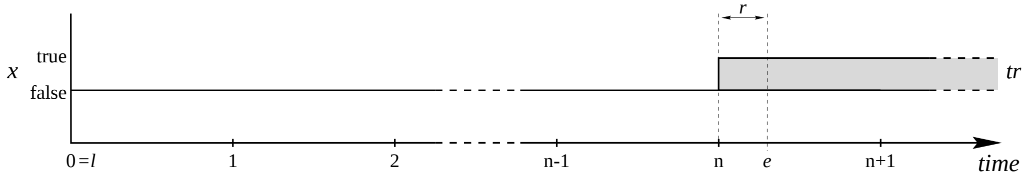
We start by showing property 1. Therefore let $k \in \mathbb{N}_{\geq 1}$ and $w \in \alpha^k$. Furthermore let tr be a time-wise discrete trajectory such that

$$\forall\, k \in \mathbb{N}_{<k} \,.\, \forall\, \delta \in (0, 1) \,.\, tr(i + \delta)(v) = w(i)(v)$$

holds for each $v \in V$. Such a $tr \in DTraj$ does obviously exist and it satisfies $w = wrd_V(tr)[0, \lfloor k - 1 + r \rfloor]$. Now either $tr, [l, k - 1 + r] \models \phi$ or $tr, [l, k - 1 + r] \models \neg\phi$ holds due to the semantics of DL. Hence, by definition of $\mathcal{L}_{l,r}$, $w \in \mathcal{L}_{l,r}(\phi)$ or $w \in \mathcal{L}_{l,r}(\neg\phi)$. As $k \in \mathbb{N}_{\geq 1}$ and $w \in \alpha^k$ were arbitrarily chosen, this shows $\mathcal{L}_{l,r}(\phi) \cup \mathcal{L}_{l,r}(\neg\phi) \supseteq \alpha^+$.

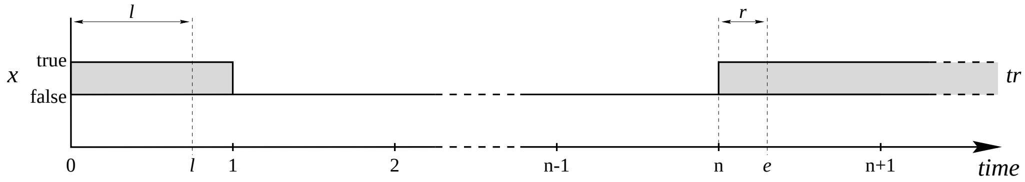
Time-wise discrete observations $(tr, [l, e])$ that satisfy $\int x < 1$, start in $l \in [0, 1)$, and end in e with offset $r \in [0, 1)$ to an integer time instant can have one of the following shapes, depending on l and r. Shaded parts indicate that both a trajectory assigning `true` to x in that phase and a trajectory assigning `false` to x in that phase satisfy $\int x < 1$.

Case 1. $0 = l \leq r < 1$.



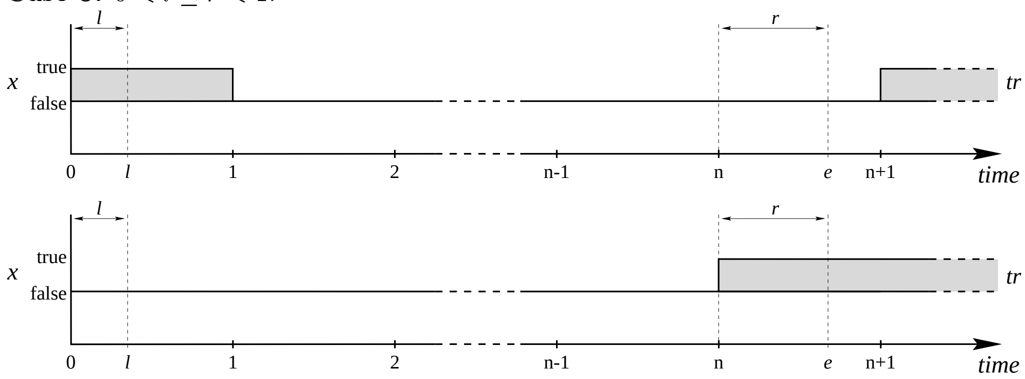
If $l = 0$ then all non-empty intersections of integer-aligned unit intervals with $[l, e]$ except the rightmost one have length 1. Hence, in order to satisfy $\int x < 1$, x has to be interpreted by `false` throughout all integer-aligned unit intervals except perhaps the rightmost, i.e. incomplete, one.

Case 2. $0 \leq r < l < 1$.



If $l > 0$ then all non-empty intersections of integer-aligned unit intervals with $[l, e]$ except the leftmost and the rightmost one have length 1. Hence, in order to satisfy $\int x < 1$, x has to be interpreted by `false` throughout all integer-aligned unit intervals except perhaps the leftmost or the rightmost one. As $r < l$, $\int x < 1$ is even satisfied if x is interpreted by `true` in both the leftmost and the rightmost integer-aligned unit interval, as then $[\![\int x]\!](tr, [l, \mathrm{n}+r]) = 1-l+r < 1$.

Case 3. $0 < l \leq r < 1$.



As in case 2, all integer-aligned unit intervals intersecting with $[l, e]$ except the leftmost and the rightmost one have length 1. Hence, x has to be interpreted by `false` throughout all integer-aligned unit intervals except perhaps the leftmost or the rightmost one. However, as $l \leq r$, $\int x < 1$ would be violated if x were interpreted by `true` in both the leftmost and the rightmost integer-aligned unit interval, as then $[\![\int x]\!](tr, [l, \mathrm{n} + r]) = 1 - l + r \geq 1$. Therefore, x may only evaluate to `true` in *either* the leftmost *or* the rightmost aligned unit interval intersecting with $[l, e]$, which gives rise to the two possible shapes of tr that are depicted above.

Figure 6.3: Time-wise discrete observations satisfying $\int x < 1$

On the other hand, $0 \leq \lfloor e \rfloor$ for any $e \in \mathit{Time}$ such that $\mathit{wrd}_V(\mathit{tr})[0, \lfloor e \rfloor] \neq \varepsilon$ for each $\mathit{tr} \in \mathit{DTraj}$ and $e \in \mathit{Time}$. Hence, $\varepsilon \notin \mathcal{L}_{l,r}(\phi)$ and $\varepsilon \notin \mathcal{L}_{l,r}(\neg\phi)$ by definition of $\mathcal{L}_{l,r}$. This implies that $\mathcal{L}_{l,r}(\phi) \cup \mathcal{L}_{l,r}(\neg\phi) \subseteq \alpha^+$. Hence, $\mathcal{L}_{l,r}(\phi) \cup \mathcal{L}_{l,r}(\neg\phi) = \alpha^+$.

We show property 2 by deriving a contradiction from its negation. Therefore assume that $\mathcal{L}_{l,r}(\phi) \cap \mathcal{L}_{l,r}(\neg\phi) \neq \emptyset$, i.e. $w \in \mathcal{L}_{l,r}(\phi) \cap \mathcal{L}_{l,r}(\neg\phi)$ exists. By definition of $\mathcal{L}_{l,r}$ this implies that $\mathit{tr}_\phi, \mathit{tr}_{\neg\phi} \in \mathit{DTraj}$ and $k_\phi, k_{\neg\phi} \in \mathbb{N}$ exist with

a) $\mathit{tr}_\phi, [l, k_\phi + r] \models \phi$,

b) $\mathit{tr}_{\neg\phi}, [l, k_{\neg\phi} + r] \models \neg\phi$, and

c) $\mathit{wrd}_V(\mathit{tr}_\phi)[0, \lfloor k_\phi + r \rfloor] = w = \mathit{wrd}_V(\mathit{tr}_{\neg\phi})[0, \lfloor k_{\neg\phi} + r \rfloor]$.

Property (c) obviously implies $k_\phi = k_{\neg\phi}$ as $k_\phi + 1 = \mathrm{len}(\mathit{wrd}_V(\mathit{tr}_\phi)[0, \lfloor k_\phi + r \rfloor]) = \mathrm{len}(w)$ and similarly $k_{\neg\phi} + 1 = \mathrm{len}(w)$. Furthermore, property (c) implies by definition of wrd_V and time-wise discreteness of tr_ϕ and $\mathit{tr}_{\neg\phi}$ that $\{t \in [0, k_\phi + 1] \mid \mathit{tr}_\phi(t) \neq \mathit{tr}_{\neg\phi}(t)\}$ is a set of measure 0. But then

$$
\begin{aligned}
& \mathit{tr}_\phi, [l, k_\phi + r] \models \phi \\
\iff\ & \mathit{tr}_{\neg\phi}, [l, k_{\neg\phi} + r] \models \phi && \text{[Lemma 6.10]} \\
\iff\ & \mathit{tr}_{\neg\phi}, [l, k_{\neg\phi} + r] \not\models \neg\phi && \text{[Semantics of DL]}
\end{aligned}
$$

which is a contradiction to properties (a) and (b). As these inconsistent properties were deduced from the single assumption $\mathcal{L}_{l,r}(\phi) \cap \mathcal{L}_{l,r}(\neg\phi) \neq \emptyset$, this assumption cannot hold, which yields property 2. Finally, properties 1 and 2 together give $\mathcal{L}_{l,r}(\neg\phi) = \alpha^+ \setminus \mathcal{L}_{l,r}(\phi)$. This completes the case $l \leq r$.

The proof for the case $l > r$ is essentially the same, the only difference being that $e = \lfloor e \rfloor + r \wedge e \geq l > r$ implies $e \geq 1$ such that then $\mathrm{len}(\mathit{wrd}_V(\mathit{tr})[0, \lfloor e \rfloor]) \geq 2$ for any $\mathit{tr} \in \mathit{DTraj}$. Hence, $\mathrm{len}(w) < 2$ implies $w \notin \mathcal{L}_{l,r}(\phi) \cup \mathcal{L}_{l,r}(\neg\phi)$, i.e. $\mathcal{L}_{l,r}(\phi) \cup \mathcal{L}_{l,r}(\neg\phi) \subseteq \alpha^{\geq 2}$. A similar argument as in the case $l \leq r$ shows that $\mathcal{L}_{l,r}(\phi) \cup \mathcal{L}_{l,r}(\neg\phi) = \{w \in \alpha^* \mid \mathrm{len}(w) \geq 2\} = \alpha^{\geq 2}$. Furthermore, $\mathcal{L}_{l,r}(\phi) \cap \mathcal{L}_{l,r}(\neg\phi) = \emptyset$ can be shown exactly as in the case $l \leq r$. Hence, $\mathcal{L}_{l,r}(\neg\phi) = \alpha^{\geq 2} \setminus \mathcal{L}_{l,r}(\phi)$ if $l > r$. ▟

Lemma 6.16. *Let ϕ_1 and ϕ_2 be formulae with $\mathit{free}(\phi_i) \subseteq V$. Then*

$$\mathcal{L}_{l,r}(\phi_1 \vee \phi_2) \quad = \quad \mathcal{L}_{l,r}(\phi_1) \cup \mathcal{L}_{l,r}(\phi_2) \ .$$

Proof.

$$
\begin{aligned}
& \mathcal{L}_{l,r}(\phi_1 \vee \phi_2) \\
= & \left\{ \mathit{wrd}_V(\mathit{tr})[0, \lfloor e \rfloor] \;\middle|\; \begin{array}{l} \mathit{tr} \in \mathit{DTraj},\ e \in \mathit{Time}, \\ e \geq l \ \wedge\ e = \lfloor e \rfloor + r, \\ \mathit{tr}, [l, e] \models \phi_1 \vee \phi_2 \end{array} \right\} && \text{[Def. 6.11]} \\
= & \left\{ \mathit{wrd}_V(\mathit{tr})[0, \lfloor e \rfloor] \;\middle|\; \begin{array}{l} \mathit{tr} \in \mathit{DTraj},\ e \in \mathit{Time}, \\ e \geq l \ \wedge\ e = \lfloor e \rfloor + r, \\ \mathit{tr}, [l, e] \models \phi_1 \ \text{ or } \ \mathit{tr}, [l, e] \models \phi_2 \end{array} \right\} && \text{[Semantics of DL]}
\end{aligned}
$$

$$= \left\{ wrd_V(tr)[0, \lfloor e \rfloor] \;\middle|\; \begin{array}{l} tr \in DTraj, \\ e \in Time, \\ e \geq l, \\ e = \lfloor e \rfloor + r, \\ tr, [l, e] \models \phi_1 \end{array} \right\} \cup \left\{ wrd_V(tr)[0, \lfloor e \rfloor] \;\middle|\; \begin{array}{l} tr \in DTraj, \\ e \in Time, \\ e \geq l, \\ e = \lfloor e \rfloor + r, \\ tr, [l, e] \models \phi_2 \end{array} \right\}$$

$$= \mathcal{L}_{l,r}(\phi_1) \cup \mathcal{L}_{l,r}(\phi_2) \qquad \text{[Def. 6.11]}$$

∎

Lemma 6.17. *Let ϕ_1 and ϕ_2 be formulae with $free(\phi_i) \subseteq V$. Then*

$$\mathcal{L}_{l,r}(\phi_1 \,;\, \phi_2) \;=\; \bigcup_{x \in [0,1)} \mathcal{L}_{l,x}(\phi_1) \hat{}\, \mathcal{L}_{x,r}(\phi_2) \ ,$$

where $X \hat{}\, Y \stackrel{\text{def}}{=} \{u \hat{}\, v \mid u \in X, v \in Y, (u,v) \in \mathrm{dom}\,(\hat{}\,)\}$ is the lifted version of concatenation with one letter overlap $\hat{}\, : \alpha^* \times \alpha^* \xrightarrow{\text{partial}} \alpha^*$ which is defined by

$$u \hat{}\, v \stackrel{\text{def}}{=} \begin{cases} u \cdot (v[0, \mathrm{len}(v) - 1]) & \text{iff} \quad u(\mathrm{len}(u) - 1) = v(0), \\ \text{undefined} & \text{iff} \quad u(\mathrm{len}(u) - 1) \neq v(0). \end{cases}$$

Note that this lemma characterizes $\mathcal{L}_{l,r}(\phi_1 \,;\, \phi_2)$ through an uncountably infinite union of languages.

Proof. First, we observe that $v[0, n] = v[0, m] \hat{}\, v[m, n]$ for each $m \leq n \leq \mathrm{len}(v) - 1$. Hence,

$$wrd_V(tr)[0, \lfloor e \rfloor] = wrd_V(tr)[0, \lfloor m \rfloor] \hat{}\, wrd_V(tr)[\lfloor m \rfloor, \lfloor e \rfloor] \tag{6.1}$$

for each $tr \in DTraj$ and $m \leq e \in Time$. Furthermore, for each $tr \in DTraj$ and $k \in \mathbb{N}$, $t \mapsto tr(t + k)$ is a time-wise discrete trajectory that satisfies

$$wrd_V(tr)[k, \lfloor e \rfloor] = wrd_V(t \mapsto wrd_V(t + k))[0, \lfloor e - k \rfloor] \ . \tag{6.2}$$

In order to show that $\mathcal{L}_{l,r}(\phi_1 \,;\, \phi_2) \subseteq \bigcup_{x \in [0,1)} \mathcal{L}_{l,x}(\phi_1) \hat{}\, \mathcal{L}_{x,r}(\phi_2)$ let $w \in \mathcal{L}_{l,r}(\phi_1 \,;\, \phi_2)$. By definition of $\mathcal{L}_{l,r}$ this implies that there is $tr \in DTraj$ and $e \in Time$ with $e = \lfloor e \rfloor + r$ such that $w = wrd_V(tr)[0, \lfloor e \rfloor]$ and $tr, [l, e] \models \phi_1 \,;\, \phi_2$. According to the semantics of chop this implies existence of $m \in Time$ with $l \leq m \leq e$ such that $tr, [l, m] \models \phi_1$ and $tr, [m, e] \models \phi_2$ hold.

Let $x \stackrel{\text{def}}{=} m - \lfloor m \rfloor$ and $k \stackrel{\text{def}}{=} \lfloor m \rfloor$. By lemma 5.1, $(t \mapsto tr(t + k)), [x, e - k] \models \phi_2$ follows from $tr, [m, e] \models \phi_2$. As k is integer, $e = \lfloor e \rfloor + r$ implies $e - k = \lfloor e - k \rfloor + r$, such that $(t \mapsto tr(t + k)), [x, e - k] \models \phi_2$ yields $wrd_V(t \mapsto tr(t + k))[0, \lfloor e - k \rfloor] \in \mathcal{L}_{x,r}(\phi_2)$ according to definition 6.11. By equation (6.2), $wrd_V(t \mapsto tr(t + k))[0, \lfloor e - k \rfloor] = wrd_V(tr)[k, \lfloor e \rfloor]$ such that

$$wrd_V(tr)[k, \lfloor e \rfloor] \in \mathcal{L}_{x,r}(\phi_2) \ . \tag{6.3}$$

Furthermore, $m = \lfloor m \rfloor + x$ and $tr, [l, m] \models \phi_1$ yield

$$wrd_V(tr)[0, \lfloor m \rfloor] \in \mathcal{L}_{l,x}(\phi_1) \tag{6.4}$$

according to definition 6.11. It follows that

$$
\begin{aligned}
& w \\
= \;& wrd_V(tr)[0, \lfloor e \rfloor] && \text{[Choice of } wrd_V \text{ and } e\text{]} \\
= \;& wrd_V(tr)[0, \lfloor m \rfloor]\hat{\ } wrd_V(tr)[\lfloor m \rfloor, \lfloor e \rfloor] && \text{[Equation (6.1)]} \\
= \;& wrd_V(tr)[0, k]\hat{\ } wrd_V(tr)[k, \lfloor e \rfloor] && [k = \lfloor m \rfloor] \\
\in \;& \mathcal{L}_{l,x}(\phi_1)\hat{\ }\mathcal{L}_{x,r}(\phi_2) && \text{[(6.3) and (6.4)]} \\
\subseteq \;& \bigcup_{y \in [0,1)} \mathcal{L}_{l,y}(\phi_1)\hat{\ }\mathcal{L}_{y,r}(\phi_2) \;. && [x = m - \lfloor m \rfloor \in [0,1)]
\end{aligned}
$$

This proves the inclusion $\mathcal{L}_{l,r}(\phi_1 \,;\, \phi_2) \subseteq \bigcup_{x \in [0,1)} \mathcal{L}_{l,x}(\phi_1)\hat{\ }\mathcal{L}_{x,r}(\phi_2)$ as $w \in \mathcal{L}_{l,r}(\phi_1 \,;\, \phi_2)$ was arbitrarily chosen. The converse inclusion follows analogously. ▟

Lemma 6.18. *Suppose P is a state assertion with free$(P) \subseteq V$ and let $k \in \mathbb{N} \setminus \{0\}$. Then*

$$
\begin{aligned}
\mathcal{L}_{l,r}(\textstyle\int P < 0) &= \mathcal{L}_{l,r}(\textstyle\int P > 0 \wedge \neg \textstyle\int P > 0) \;, \\
\mathcal{L}_{l,r}(\textstyle\int P > 1) &= \mathcal{L}_{l,r}(\neg(\textstyle\int P < 1) \,;\, \textstyle\int P > 0) \;, \\
\mathcal{L}_{l,r}(\textstyle\int P < k+1) &= \mathcal{L}_{l,r}(\textstyle\int P < k \,;\, \textstyle\int P < 1) \;, \\
\mathcal{L}_{l,r}(\textstyle\int P > k+1) &= \mathcal{L}_{l,r}(\textstyle\int P > k \,;\, \textstyle\int P > 1) \;.
\end{aligned}
$$

Note that this characterization allows to determine $\mathcal{L}_{l,r}(\int P < k)$ and $\mathcal{L}_{l,r}(\int P > k)$ for arbitrary $k \in \mathbb{N}$ from the characterizations of $\mathcal{L}_{l,r}(\int P < 1)$ and $\mathcal{L}_{l,r}(\int P > 0)$ given in lemmata 6.13 and 6.14.

Proof. From definition 6.11 it is obvious that $\mathcal{L}_{l,r}(\phi) = \mathcal{L}_{l,r}(\psi)$ if ϕ and ψ are logically equivalent. Hence, above equations follow from the logical equivalences observed in section 5.1.5. ▟

Lemmata 6.14 to 6.18 provide a recursive — except for lemma 6.18 even structurally inductive — characterization of $\mathcal{L}_{l,r}(\phi)$ for arbitrary $l, r \in [0,1)$ and DL formulae ϕ with *free*$(\phi) \subseteq V$. For the base cases, lemmata 6.13 and 6.14 even provide regular expressions such that $\mathcal{L}_{l,r}(\phi)$ can be seen to be effectively constructible in the base cases. Negation and disjunction preserve effectiveness, as they map to language difference and language union (lemmata 6.15 and 6.16), which are both effective in the realm of regular languages. Unfortunately, lemma 6.17, which deals with chop, involves an uncountably infinite union of languages such that it is not immediately clear whether chop preserves regularity.

Fortunately, this particular infinite union of languages consists of only finitely many different constituents according to the following lemma, such that regularity of the language $\mathcal{L}_{l,r}(\phi)$ is also preserved under chop.

Lemma 6.19 (Finiteness of the family of languages $(\mathcal{L}_{l,r}(\phi))_{l,r \in [0,1)}$). *Let $l_1, l_2, r_1, r_2 \in [0,1)$ and let ϕ be a DL formula such that free$(\phi) \subseteq V$. If*

(i) $0 = l_1 = r_1$ *and* $0 = l_2 = r_2$ *or*

(ii) $0 = l_1 < r_1$ *and* $0 = l_2 < r_2$ *or*

(iii) $0 < l_1 < r_1$ *and* $0 < l_2 < r_2$ *or*

(iv) $0 < l_1 = r_1$ *and* $0 < l_2 = r_2$ *or*

(v) $0 = r_1 < l_1$ *and* $0 = r_2 < l_2$ *or*

(vi) $0 < r_1 < l_1$ *and* $0 < r_2 < l_2$

then $\mathcal{L}_{l_1,r_1}(\phi) = \mathcal{L}_{l_2,r_2}(\phi)$ *holds. I.e.* $\mathcal{L}_{l,r}(\phi)$ *depends only on* $\mathrm{sgn}(l)$, $\mathrm{sgn}(r)$, *and* $\mathrm{sgn}(l-r)$ *rather than on the exact values of* l *and* r. *Therefore, the family of languages* $(\mathcal{L}_{l,r}(\phi))_{l,r\in[0,1)}$ *contains at most six different languages for each* ϕ.

Proof. This property follows through a straightforward induction from the characterizations of $\mathcal{L}_{l,r}(\phi)$ provided by lemmata 6.14 to 6.18. Note that the case distinction (i) to (vi) is indeed taken from the intersection of the case distinctions made in those lemmata such that the proposition can immediately be seen to hold for atomic formulae ϕ of the forms $\phi = \int P < 1$ or $\phi = \int P > 0$ and to be preserved by the connectives $\neg$, $\vee$, and ;. Hence, structural induction reveals that the proposition is true for all formulae built from atomic formulae of the forms $\phi = \int P < 1$ and $\phi = \int P > 0$ using Boolean connectives and chop. Finally, lemma 6.18 shows that for each formula ϕ built from the full set $\{\int P < k, \int P > k \mid P \text{ is a state assertion}, k \in \mathbb{N}\}$ of atomic formulae using Boolean connectives and chop there is a formula $\bar{\phi}$ that contains only atomic formulae of the restricted forms $\phi = \int P < 1$ and $\phi = \int P > 0$ and satisfies $\mathcal{L}_{l,r}(\bar{\phi}) = \mathcal{L}_{l,r}(\phi)$ for each $l, r \in [0,1)$. Thus, the proposition holds for arbitrary formulae of Duration Logic. ∎

As this lemma does not only show that the apparently uncountable union of languages in lemma 6.17 consists of only finitely many different constituents, but furthermore provides an effective reduction to such a finite union, it shows that beyond regularity of the language even effective constructibility of the corresponding regular expression is preserved under chop. Thus, all operators of DL preserve effectiveness of the construction:

Corollary 6.20. $\mathcal{L}_{l,r}(\phi)$ *is regular and can be effectively constructed for each* $l, r \in [0,1)$ *and each DL formula* ϕ *with free*$(\phi) \subseteq V$. ∎

Furthermore, lemma 6.19 shows that $\bigcup_{r\in[0,1)} \mathcal{L}_{0,r}(\phi)$ is a union of at most two different languages, as $\mathcal{L}_{0,r_1}(\phi) = \mathcal{L}_{0,r_2}(\phi)$ for any $r_1, r_2 \in (0,1)$. As these two languages are effectively constructible regular languages due to corollary 6.20, this implies

Corollary 6.21. $\bigcup_{r\in[0,1)} \mathcal{L}_{0,r}(\phi)$ *is an effectively constructible regular set for each formula* ϕ *with free*$(\phi) \subseteq V$. ∎

As emptiness of regular languages is decidable, this implies

Corollary 6.22. *It is decidable whether* $\bigcup_{r\in[0,1)} \mathcal{L}_{0,r}(\phi) = \emptyset$ *for each formula* ϕ *of DL with free*$(\phi) \subseteq V$. ∎

By lemma 6.12 this implies decidability of the validity problem over time-wise discrete models.

Corollary 6.23. *For each formula ϕ of DL with $free(\phi) \subseteq V$ it is decidable whether ϕ is valid on time-wise discrete models.*

Proof. According to lemma 6.12, ϕ is valid iff $\bigcup_{r \in [0,1)} \mathcal{L}_{0,r}(\neg\phi) = \emptyset$. By corollary 6.22, applied to $\neg\phi$ instead of ϕ, this is decidable. ∎

As the set V was arbitrarily chosen, this shows

Theorem 6.24 (Decidability of DL over time-wise discrete models). *Validity over time-wise discrete models is decidable for formulae of DL.*

Proof. Given an arbitrary formula ϕ, the construction of lemmata 6.14 to 6.19 can be performed with $V = free(\phi)$. Corollary 6.23 then shows that validity of ϕ over time-wise discrete models is decidable. ∎

An analysis of the extended regular expressions that the construction of lemmata 6.14 to 6.19 assigns to formulae reveals that this particular decision procedure for validity over time-wise discrete models inherits non-elementary complexity from the language emptiness problem for extended regular expressions [MS73]. Furthermore, it is easy to see that in general one cannot do better:

Lemma 6.25. *The validity problem of Duration Logic over time-wise discrete models is non-elementary.*

Proof. The emptiness problem of extended regular expressions, which is non-elementary according to [MS73], has a linear encoding in DL: let e be an extended regular expression over an alphabet of the form $\alpha = V \to \mathbb{B}$, with $V \subset Varname$ being finite. Assume, for the moment, that $clk \in Varname \setminus V$ is a DL variable which changes state at and only at *every* integer time instant, i.e. represents a digital oscillator of period 2. Then, the DL formula $F(e)$ defined by

$$\begin{aligned}
F(a) &\stackrel{\text{def}}{=} \lceil P \rceil^1 \quad \text{if } a \in \alpha \text{ and } P \text{ is a state assertion with } \widetilde{P} = \{a\}, \\
F(e_1 \vee e_2) &\stackrel{\text{def}}{=} F(e_1) \vee F(e_2) \ , \\
F(e_1 \cdot e_2) &\stackrel{\text{def}}{=} F(e_1) \,;\, F(e_2) \ , \\
F(\neg e_1) &\stackrel{\text{def}}{=} \neg F(e_1) \wedge \left(\lceil\,\rceil \vee \mathtt{true} \,;\, \lceil clk \rceil^1 \vee \mathtt{true} \,;\, \lceil \neg clk \rceil^1 \right)
\end{aligned}$$

is finitely satisfied by a time-wise discrete trajectory tr on an observation interval $[0, e]$ if and only if $e \in \mathbb{N}$ and $wrd_V(tr)[0, e] \in \mathcal{L}_e$, where $\mathcal{L}_e$ is the regular language denoted by e. Thus, $\neg F(e)$ is satisfied by all time-wise discrete trajectories that meet the given assumption on clk iff $\mathcal{L}_e$ is empty.

Now, given that we deal with time-wise discrete models only, a digital oscillator *clk* of period 2 can be formalized by the DL formula

$$\mathit{Clock} \stackrel{\text{def}}{=} \Big(\lceil clk \rceil^1 \rightsquigarrow \lceil \neg clk \rceil\Big) \wedge \Big(\lceil \neg clk \rceil^1 \rightsquigarrow \lceil clk \rceil\Big) \ .$$

Hence, $\phi_e \stackrel{\text{def}}{=} (\mathit{Clock} \Rightarrow \neg F(e))$ is a DL formula that is valid on time-wise discrete models if and only if $\mathcal{L}_e$ is empty. This provides a linear encoding of the emptiness problem of extended regular expressions in DL. ∎

However, the non-elementary complexity of the decision procedure need not be an obstacle to practical applications, as deep nesting of chop and negation which leads to the blow-up in complexity is hardly ever encountered in practice, except within iterated application of some derived standard operators. In most cases, more efficient decision procedures than obtained through unfolding their definitions can be easily devised for these derived operators. E.g. the frequently encountered timed leads-to properties $\lceil P \rceil^k \rightsquigarrow \lceil Q \rceil$ can be easily checked through cycle counting, testing whether every consecutive sequence of k letters from $\widetilde{P}$ is followed by a letter from $\widetilde{Q}$, whereas unfolding the definitions of $\rightsquigarrow$, $\lceil \cdot \rceil$, and $\cdot^k$ given in section 5.1.5 yields a formula of frightening size with some nested negations and chops, thus providing a burden to the basic decision procedure. While such a special treatment of derived operators does not influence the worst-case complexity, it does reduce complexity of checking formulae where deep nesting of negation and chop *only* occurs through iterated application of derived operators. In a prototypic implementation of the decision procedure performed by the author, enhanced treatment of timed leads-to and some other frequently encountered derived operators resulted in a dramatic increase of performance when applied to the gas-burner example [RRH93].

6.4 Decidability of other Duration Calculi on restricted model classes

A bunch of work on decidability of subsets of Duration Calculus has been reported by Zhou Chaochen, M. R. Hansen, and P. Sestoft in [ZHS93]. In order to allow comparison of subset DL of Duration Calculus with other possible choices, we report their results in this section and extend them to cover also the model classes 'n-bounded' and 'time-wise discrete', which Zhou Chaochen, M. R. Hansen, and P. Sestoft did not study. An overview of the results, covering also those of the previous three sections, is given in table 6.1 on page 57.

6.4.1 Subset $\{\lceil P \rceil\}$

This subset consists of the formulae that can be constructed from primitive formulae of the form $\lceil \langle \mathit{state\,assertion} \rangle \rceil$ using Boolean connectives and chop, where $\lceil P \rceil$ means that the observation interval has positive length and that P holds almost everywhere in the observation interval. Note that this subset has neither a notion of accumulated duration nor any *metric* notion of time.

Finitely variable models

According to [ZHS93, Theorem 10], the $\{\lceil P \rceil\}$-fragment of Duration Calculus is decidable over finitely variable models.

n-bounded models

As fragment $\{\lceil P \rceil\}$ is a subset of fragment $\{\lceil P \rceil, l = k\}$, positive decidability results for the latter subset cover the former also. We will see in section 6.4.2 that subset $\{\lceil P \rceil, l = k\}$ is decidable over n-bounded models for each $n \in \mathbb{N}_{\geq 1}$. This implies

Corollary 6.26. *Validity of $\{\lceil P \rceil\}$-formulae is decidable over n-bounded models for each $n \in \mathbb{N}_{\geq 1}$.*

Proof. According to lemma 6.29 decidability holds for the even larger subset $\{\lceil P \rceil, l = k\}$. ∎

Time-wise discrete models

Via the encoding $\lceil P \rceil \mathrel{\hat{=}} \int \mathtt{true} > 0 \land \neg(\int \neg P > 0)$, subset $\{\lceil P \rceil\}$ is contained in DL. Hence, theorem 6.24 implies decidability of the $\{\lceil P \rceil\}$-fragment over time-wise discrete models.

Corollary 6.27. *Validity of $\{\lceil P \rceil\}$-formulae is decidable over time-wise discrete models.* ∎

6.4.2 Subset $\{\lceil P \rceil, l = k\}$

The $\{\lceil P \rceil, l = k\}$-fragment of Duration Calculus extends the previous subset by adding a second kind of primitive formulae: $l = k$. This allows to express the full range of integer constraints on the length of the observation interval using the encodings

$$\begin{aligned} l > k &\stackrel{\text{def}}{=} l = k \,;\, \neg\, l = 0 \ , \\ l \geq k &\stackrel{\text{def}}{=} l = k \lor l > k \ , \\ l < k &\stackrel{\text{def}}{=} \neg\, l \geq k \ , \\ l \leq k &\stackrel{\text{def}}{=} \neg\, l > k \ . \end{aligned}$$

Hence, the $\{\lceil P \rceil, l = k\}$-fragment has a quite expressive vocabulary for metric time properties, but still lacks a notion of accumulated duration. As can be seen from the abbreviations introduced in section 5.1.5, the $\{\lceil P \rceil, l = k\}$-fragment is a subset of $\{\int P < k, \int P > k\}$. It is even a proper subset, as $\{\int P < k, \int P > k\}$ features accumulated durations.

Finitely variable models

Undecidability of the $\{\lceil P \rceil, l = k\}$-fragment of Duration Calculus over finitely variable models has been shown by Zhou Chaochen, M. R. Hansen, and P. Sestoft in theorem 11 of [ZHS93].

Nevertheless, we may note for future use an effectiveness result concerning $\{\lceil P \rceil, l = k\}$-formulae that do not contain negation at the formula level (but possibly within state assertions). As neither conjunction nor interval length properties other than $l = k$ can be expressed in $\{\lceil P \rceil, l = k\}$ without negation at formula level, these constructs have to be included explicitly: the negation-free $\{\lceil P \rceil, l < k, l = k, l > k\}$-subset of Duration Calculus is the set of all formulae built from atomic formulae of the forms $\lceil P \rceil$, $l < k$, $l = k$, and $l > k$ using conjunction, disjunction, and chop.

Lemma 6.28 (Effectiveness of negation-free $\{\lceil P \rceil, l < k, l = k, l > k\}$-DC). *The left-continuous finite models of negation-free $\{\lceil P \rceil, l < k, l = k, l > k\}$-formulae form regular observation sets. Furthermore, these observation sets can be effectively constructed from the syntactic structure of the formula.*

Proof. It is straightforward to map formulae of the $\{\lceil P \rceil, l < k, l = k, l > k\}$-subset that do not contain negation to timed regular expressions describing their left-continuous[3] finite models of finite variability. An appropriate mapping R of formulae ϕ with $\mathit{free}(\phi) \subseteq V$, where $V \subseteq \mathit{Varname}$ is arbitrarily chosen, to a timed regular expression over $\alpha \stackrel{\text{def}}{=} V \to \mathbb{B}$ is

$$
\begin{aligned}
R(\lceil P \rceil) &\stackrel{\text{def}}{=} \left(\left(\bigvee\nolimits_{p \in \widetilde{P}} p\right)^*\right)_{(0,\infty)} \ , \\
R(l < k) &\stackrel{\text{def}}{=} (\alpha^*)_{[0,k)} \ , \\
R(l = k) &\stackrel{\text{def}}{=} (\alpha^*)_{[k,k]} \ , \\
R(l > k) &\stackrel{\text{def}}{=} (\alpha^*)_{(k,\infty)} \ , \\
R(\phi \wedge \psi) &\stackrel{\text{def}}{=} R(\phi) \wedge R(\psi) \ , \\
R(\phi \vee \psi) &\stackrel{\text{def}}{=} R(\phi) \vee R(\psi) \ , \\
R(\phi \,;\, \psi) &\stackrel{\text{def}}{=} R(\phi) \cdot R(\psi) \ ,
\end{aligned}
$$

where $\widetilde{P}$ is defined as in section 6.3, i.e. denotes the set of valuations in the alphabet α that satisfy P. Hence, the left-continuous, finitely variable observations satisfying some negation-free $\{\lceil P \rceil, l < k, l = k, l > k\}$-formula form an effectively constructible regular observation set. ∎

Note that this implies that finite satisfiability of negation-free $\{\lceil P \rceil, l < k, l = k, l > k\}$-formulae and thus validity of their negations is decidable, as emptiness of timed regular languages is decidable according to corollary 4.3.

n-bounded models

Decidability of the $\{\lceil P \rceil, l = k\}$-subset over n-bounded models has not been dealt with in the literature. However, it is simple to devise an appropriate decision procedure:

[3] As timed regular expressions have been defined to have left-continuous models.

Theorem 6.29. *Validity of formula of the $\{\lceil P \rceil, l = k\}$-fragment of Duration Calculus over n-bounded models is decidable for each $n \geq 1$.*

Proof. By lemma 6.28, the left-continuous finite models of finite variability of any negation-free formula of the $\{\lceil P \rceil, l = k\}$-subset form an effectively constructible regular observation set. The construction takes advantage of an effectively constructible timed regular expression $R(\phi)$ that represents the left-continuous finite models of the negation-free formula ϕ. Furthermore, for any $n \in \mathbb{N}_{\geq 1}$,

$$n\text{-bounded} \stackrel{\text{def}}{=} \bigvee_{i \in \mathbb{N}_{<n}} \alpha^i \vee \bigwedge_{i \in \mathbb{N}_{<n}} \left(\alpha^i \cdot \left((\alpha^n)_{[1,\infty)} \right)^* \cdot \alpha^{n-i} \right)$$

is a timed regular expression that represents exactly the left-continuous observations of n-bounded variability. Hence, $(R(\phi) \wedge n\text{-bounded})$ is an — by effectiveness of timed regular expressions — effective representation of ϕ's n-bounded, left-continuous finite models. What remains to be dealt with is negation, which is not covered by the definition of R.

According to [Wil94b, Corollary 5], the timed regular languages of n-bounded variability form an effective Boolean algebra. Unfortunately, T. Wilke's notion of a timed regular language is different from that used in section 4.5: Wilke's interpretation of timed automata is only concerned with infinite traces (which, except for the possibility of multiple occurrences of the same symbol in a row, correspond to trajectories in our terminology), whereas timed regular expressions are interpreted by finite observations. However, as lemma 6.30 below shows, corollary 5 of [Wil94b] can be transferred to regular observation sets of n-bounded variability, then using complement relative to the left-continuous finite observations of n-bounded variability. Hence, complementation within the left-continuous observations of n-bounded variability can be done effectively for regular observation sets. Using this complementation operation, R can be extended to cover negation also, which provides an effective representation of the left-continuous, n-bounded finite models of all $\{\lceil P \rceil, l = k\}$-formulae. By decidability of the emptiness problem of timed regular languages [AD94, Theorem 4.16] it is thus decidable whether a formula ϕ of fragment $\{\lceil P \rceil, l = k\}$ is finitely satisfiable by a left-continuous, n-bounded model. As any n-bounded observation is almost equal to a left-continuous one, this together with lemma 5.1 implies decidability of finite satisfiability of ϕ by an n-bounded model, which is equivalent to deciding validity of $\neg\phi$ over n-bounded models. ∎

Lemma 6.30. *The regular observation sets of n-bounded variability form an effective Boolean algebra (for any $n \in \mathbb{N}$).*

Sketch of proof. As the union and intersection operators of timed regular expressions (which are one of the possible representations of regular observation sets) correspond to their set-theoretic counterparts, only negation remains to be dealt with.

The left-continuous, n-bounded observations are in one-to-one correspondence to the finite timed traces that

1. do not contain the same letter twice in a row, and

2. do not have more than n time stamps belong to the same unit interval of *Time*.

These properties are easily definable by timed automata. Thus, in particular, it would suffice to have an effective complementation operation wrt. the regular sets of finite timed traces that satisfy property 2, as property 1 can be enforced through an automaton product after complementation has been performed wrt. the larger set of finite traces satisfying 2.

In [Wil94b, Corollary 5], T. Wilke has shown that complementation is effective for regular sets of *infinite* timed traces satisfying property 2. However, any *finite* timed trace can be extended to an infinite trace over an extended alphabet by appending an infinite sequence of communications of a special symbol $\mathtt{idle} \notin \alpha$. This provides an embedding of the sets of finite timed traces into those of infinite timed traces that obviously preserves regularity. Therefore, a regular set of finite timed traces can be effectively complemented by first using this embedding, than doing complementation wrt. the infinite timed traces satisfying property 2, and then restricting (through an automaton product) attention to those traces in the complement that are in the image of the embedding, i.e. those traces that end in an infinite sequence of `idle` and do not alternate between `idle` communications and communications from α. ∎

Time-wise discrete models

Like the $\{\lceil P \rceil\}$-subset, the $\{\lceil P \rceil, l = k\}$-fragment is covered by subset $\{\int P < k, \int P > k\}$, using the encodings $\lceil P \rceil \mathrel{\hat{=}} \int\mathtt{true} > 0 \wedge \neg(\int \neg P > 0)$ and $l = k \mathrel{\hat{=}} \neg(\int\mathtt{true} < k \vee \int\mathtt{true} > k)$. Thus, theorem 6.24 implies decidability of subset $\{\lceil P \rceil, l = k\}$ over time-wise discrete models.

Corollary 6.31. *Validity of $\{\lceil P \rceil, l = k\}$-formulae is decidable over time-wise discrete models.* ∎

Note that even for this subset the complexity of deciding the time-wise discrete validity problem is non-elementary, as the construction in the proof of lemma 6.25 refers to formulae of the $\{\lceil P \rceil, l = k\}$-fragment only. Hence, the emptiness problem of extended regular expressions can be linearly encoded as a time-wise discrete validity problem within this subset.

6.4.3 Subset $\{\int P = \int Q\}$

Formulae of this fragment are built from primitive formulae of the form $\int\langle \mathit{state\,assertion}\rangle = \int\langle \mathit{state} - \mathit{assertion}\rangle$ using Boolean connectives and chop. This subset covers the $\{\lceil P \rceil\}$-fragment through the encoding

$$\lceil P \rceil \quad \stackrel{\text{def}}{=} \quad \int P = \int\mathtt{true} \wedge \neg \int\mathtt{true} = \int\mathtt{false} \ ,$$

where the conjunct $\neg \int\mathtt{true} = \int\mathtt{false}$ enforces that the observation interval is a non-point interval, i.e. has positive length. However, subset $\{\int P = \int Q\}$ fails to express metric properties of time and is thus incomparable to the $\{\lceil P \rceil, l = k\}$-fragment and the $\{\int P < k, \int P > k\}$-fragment.

According to theorem 12 of [ZHS93], the $\{\int P = \int Q\}$-fragment of Duration Calculus is undecidable over finitely variable models. But the proof of this result is indeed much stronger, revealing also undecidability over n-bounded as well as over time-wise discrete models. The encoding of the halting problem for two-counter machines used in the proof of [ZHS93, Theorem 12] can be seen to work not only over finitely variable models and for discrete-time Duration Calculus, as stated in [ZHS93], but also for n-bounded or even time-wise discrete models over dense time:

Observation 6.32. *The proof of theorem 12 of [ZHS93] shows that validity of $\{\int P = \int Q\}$-formulae is undecidable over n-bounded (for any $n \in \mathbb{N}_{\geq 1}$) as well as over time-wise discrete models.* ●

Note that the undecidability result for the $\{\int P = \int Q\}$-subset over n-bounded models could, for $n \geq 2$, also be shown using the "slowing down"-technique of section 6.2, as the proof of the base case of lemma 6.3 provides enough information to easily accommodate formulae of the form $\int P = \int Q$ also.

6.4.4 Subset $\{\lceil P \rceil, \forall x, l = x\}$

This fragment is built from primitive formulae of the two forms $\lceil \langle \mathit{state\,assertion} \rangle \rceil$ and $l = \langle \mathit{rigid\,variable} \rangle$ using Boolean connectives, chop, and quantification over rigid variables. Rigid variables are interpreted by real-valued valuations. Having first-order quantification, this subset is not contained in any of the other subsets. Vice versa, not featuring accumulated durations it does not enclose $\{\int P = \int Q\}$ and $\{\int P < k, \int P > k\}$, thus being incomparable to those subsets. But $\{\lceil P \rceil, \forall x, l = x\}$ obviously contains the $\{\lceil P \rceil\}$-fragment and, using rigid variables to represent the time unit and the constant 0,[4] the $\{\lceil P \rceil, l = k\}$-fragment.

According to theorem 13 of [ZHS93], the $\{\lceil P \rceil, \forall x, l = x\}$-fragment of Duration Calculus is undecidable over finitely variable models. But as before, the proof of the theorem implies even undecidability over n-bounded as well as over time-wise discrete models. The particular encoding of the halting problem for two-counter machines used in the proof of theorem 13 of [ZHS93] can again be seen to work also for n-bounded and time-wise discrete models:

Observation 6.33. *The proof of theorem 13 of [ZHS93] shows that validity of formulae in the fragment $\{\lceil P \rceil, \forall x, l = x\}$ is undecidable over n-bounded (for any $n \in \mathbb{N}_{\geq 1}$) as well as over time-wise discrete models.* ●

[4] The trick is to replace $l = k$ by the k-fold chop of the formula $l = \mathit{unit}$ if $k > 0$ and by $l = \mathit{zero}$ if $k = 0$ and to enclose the resulting formula ϕ into the context

$$\exists\, \mathit{unit}\,.\, \exists\, \mathit{zero}\,.\, \Box(l = \mathit{unit} \Rightarrow \lceil \mathtt{true} \rceil) \wedge \Box(l = \mathit{zero} \Rightarrow \neg \lceil \mathtt{true} \rceil) \wedge \phi \quad ,$$

where existential quantification is defined as the dual of universal quantification and $\Box\cdot$ is defined as in section 5.1.5. The conjunct $\Box(l = \mathit{unit} \Rightarrow \lceil \mathtt{true} \rceil)$ ensures that the otherwise arbitrarily chosen time unit is non-zero, and the conjunct $\Box(l = \mathit{zero} \Rightarrow \neg \lceil \mathtt{true} \rceil)$ ensures that rigid variable zero represents the constant 0.

		Model class			
		finitely variable	n-bounded, $n \in \mathbb{N}_{\geq 2}$	1-bounded	time-wise discrete
Subset of DC	$\{\lceil P \rceil\}$	Decidable *[ZHS93, Th. 10]*	Decidable *Corollary 6.26*	Decidable *Corollary 6.26*	Decidable *Corollary 6.27*
	$\{\lceil P \rceil, l = k\}$	Undecidable *[ZHS93, Th. 11]*	Decidable *Theorem 6.29*	Decidable *Theorem 6.29*	Decidable *Corollary 6.31*
	$\{\int P < k, \int P > k\}$	Undecidable *Theorem 6.1*	Undecidable *Theorem 6.7*	Undecidable *Theorem A.7*	Decidable *Theorem 6.24*
	$\{\int P = \int Q\}$	Undecidable *[ZHS93, Th. 12]*	Undecidable *Observ. 6.32*	Undecidable *Observ. 6.32*	Undecidable *Observ. 6.32*
	$\{\lceil P \rceil, \forall x, l = x\}$	Undecidable *[ZHS93, Th. 13]*	Undecidable *Observ. 6.33*	Undecidable *Observ. 6.33*	Undecidable *Observ. 6.33*

Table 6.1: Decidability of fragments of Duration Calculus over different model classes

6.5 Discussion

In this chapter we have investigated decidability of Duration Logic DL, i.e. decidability of the $\{\int P < k, \int P > k\}$-subset of full Duration Calculus, as well as decidability of other fragments of Duration Calculus that are found in the literature. This completes the picture provided by Zhou Chaochen, M. R. Hansen, and P. Sestoft in [ZHS93]. An overview of the results is presented in table 6.1.

The results indicate that decidability over finitely variable as well as over n-bounded models is hard to achieve for a temporal logic featuring a notion of accumulated durations. This is not surprising, given the close correspondence of accumulated durations to stopwatches in timed transition tables and the fact that introduction of even a single stopwatch destroys decidability of reachability problems of timed transition tables [HKPV95, Theorem 4.1]. The only subset of Duration Calculus that is known to be decidable over finitely variable models is $\{\lceil P \rceil\}$ which neither has a notion of accumulated duration ($\lceil P \rceil$ just being a formula expressing an invariance property of the state over the observation interval) nor a means to express metric properties of time. The largest subset that was found to be decidable on n-bounded models is $\{\lceil P \rceil, l = k\}$, which still is no genuine *Duration* Calculus as it lacks a notion of accumulated duration. Among the two studied subsets $\{\int P < k, \int P > k\}$ and $\{\int P = \int Q\}$ that feature a notion of accumulated duration, subset $\{\int P < k, \int P > k\}$, i.e. DL, is the more well-behaved as it is decidable over time-wise discrete models. This coincides nicely with the fact that its vocabulary is also more well-suited to specification of real-time systems, covering e.g. the vocabulary needed for the gas-burner design [RRH93]. In contrast, subset $\{\int P = \int Q\}$ fails to express any metric real-time properties. Given that trajectories evolving discretely over time are naturally considered when designing synchronously clocked real-time controllers, as M. R. Hansen and E.-R. Olderog convincingly argue in [HO93, Han94b], the positive decidability result on time-wise discrete models and the adequate vocabulary give us a double reason for selecting fragment $\{\int P < k, \int P > k\}$, i.e. DL, as the specification language used throughout the rest of this thesis.

The idea of restricting attention to trajectories that change only at evenly spaced time instants has also been explored in [ZHS93, HO93, SS94, Han94b]. But the changes that have been made to the semantics of Duration Calculus in that line of research have been more drastic than in our setting: instead of just reducing the model class, Zhou Chaochen, M. R. Hansen, P. Sestoft, and J.-U. Skakkebæk have studied interpretation of Duration Calculus over a *discrete time domain*.

Interpretation of Duration Calculus over a discrete time domain, which was selected to be the non-negative integers $\mathbb{N}$, requires some straightforward modifications to the semantics of Duration Calculus: first, the time domain *Time* is now defined to be $\mathbb{N}$, which implicitly changes observation intervals $[a, b] \subset \mathit{Time}$ to be integer subranges and trajectories $tr \in \mathit{Traj} \stackrel{\text{def}}{=} \{tr : \mathit{Time} \to \Sigma \mid tr \text{ finitely variable}\}$ to be Boolean valuations of state variables over integer time points.[5] Second, the semantics of durations has to be adapted to use summation instead of integration, yielding

$$[\![\textstyle\int P]\!](tr, [a, b]) \stackrel{\text{def}}{=} \sum_{t=a}^{b-1} (\chi \circ [\![P]\!] \circ tr(t)) \quad .$$

The decidability results reported in [ZHS93] are as follows.

Subset of Duration Calculus	$\lceil P \rceil$	$\lceil P \rceil, l = k$	$\int P = \int Q$	$\lceil P \rceil, \forall x, l = x$
Decidability over discrete time	Decidable	Decidable	Undecidable	Undecidable
Theorem no. in [ZHS93]	3	4	12	13

This seems encouraging also with respect to Duration Logic, as in the discrete time setting, the formula $\int P > k$ can be encoded in the $\{\lceil P \rceil, l = k\}$-fragment as

$$\mathtt{true}\,;\lceil P \rceil\,;\mathtt{true}\,;\lceil P \rceil\,;\mathtt{true}\,;\ldots\,;\mathtt{true}\,;\lceil P \rceil\,;\mathtt{true}$$

with $k+1$ occurrences of $\lceil P \rceil$ and the formula $\int P < k$ can be encoded as

$$\neg(\mathtt{true}\,;\lceil P \rceil\,;\mathtt{true}\,;\lceil P \rceil\,;\mathtt{true}\,;\ldots\,;\mathtt{true}\,;\lceil P \rceil\,;\mathtt{true})$$

with k occurrences of $\lceil P \rceil$. This shows that the decidability result of [ZHS93, Theorem 4] covers the discrete-time interpretation of the $\{\int P < k, \int P > k\}$-fragment also.

[5] Clearly, the explicit restriction of trajectories to finite variability becomes irrelevant with a discrete time domain.

However, the change of the timing domain from dense to discrete time comes at a price. As chop involves quantification over time due to the semantic rule

$$tr, [a,b] \models (\phi ; \psi) \quad \stackrel{\text{def}}{\Longleftrightarrow} \quad \exists\, m \in \mathit{Time} \,.\, \begin{pmatrix} a \leq m \leq b & \text{and} \\ tr, [a,m] \models \phi & \text{and} \\ tr, [m,b] \models \psi & \end{pmatrix} \;,$$

exchanging a densely ordered time domain against a discretely ordered one significantly changes the semantics of chop. A consequence is that the notions of validity in dense-time and in discrete-time Duration Calculus are not closely related. Validity of a formula ϕ in discrete time Duration Calculus is *neither a necessary nor a sufficient condition* for validity of ϕ in the dense time interpretation. Examples are the formulae $l > 0 \,;\, l > 0 \Rightarrow l \geq 2$, which is valid in discrete-time Duration Calculus and invalid in dense-time Duration Calculus, and $l > 0 \Rightarrow l > 0 \,;\, l > 0$, which is invalid in discrete-time Duration Calculus, but valid in dense-time Duration Calculus. Thus, discrete-time Duration Calculus does not interface smoothly to the large amount of research that has been invested in stepwise system design from dense-time Duration Calculus. Essentially, discrete-time Duration Calculus is to be considered a calculus of its own such that its decidability results are meaningless in (dense-time) Duration Calculus-based design paradigms like the ProCoS tower of abstractions [B$^+$93, BFOR93, HHF$^+$94, BHH$^+$94, HHMO$^+$96].

The lesson we learned from this semantic inconsistency was to follow M. R. Hansen's observation that designing synchronously clocked controllers imposes a natural digitization of the time domain, yet implement less drastic changes to DL's semantics than discrete-time DC. Like discrete-time Duration Calculus we restrict trajectories to change only in integer time points, but do not enforce this by replacing the time domain itself. Instead we studied validity of DL formulae over *dense time* when the model class is restricted to *time-wise discrete trajectories*, giving a notion of validity that is clearly related to the standard notion of validity over finitely variable models. Validity of a formula ϕ over time-wise discrete models is a necessary, yet not sufficient, condition for validity of ϕ.

Theorem 6.24 states that the $\{\int P < k, \int P > k\}$-fragment of Duration Calculus — as case-studies indicate the most useful of the studied subsets — is decidable over time-wise discrete models. As far as we know, this result is the first positive decidability result for a logic featuring dense-time chop, a notion of accumulated duration, and unrestricted negation. Positive decidability results found in the literature are either based on a discrete-time interpretation [SS94, Han94b] or restrict the syntax of Duration Calculus even further, omitting accumulated durations and metric time [ZHS93, Theorem 10], omitting chop [ZZYL94], or restricting the use of negation [BL95, Lak96]. However, theorem 6.24 relies on restriction of the model class to time-wise discrete models, which is only natural within *fully* synchronous systems; a situation hardly encountered in embedded control where the environment evolves independently. Therefore we will have to find a means to safely use the decidability results even in presence of unsynchronized inputs, which will be the theme of chapter 8.

7

Automatic Verification and Automatic Synthesis of Controller Designs

Duration Logic was designed for bridging the gap between requirements capture and controller implementation in an embedded-controller design activity. Consequently, we are in practice not so much interested in the tautologies of Duration Logic, i.e. formulae that are satisfied by any possible trajectory, but in the satisfaction problem between controllers from a certain design space and requirements expressed in Duration Logic. The goal for automation then becomes model checking rather than recognizing the DL tautologies, the theme of the previous chapter. However, these problems are traditionally deemed to be closely related. A controller design is usually thought to be correct if and only if each possible physical behaviour of the designed device satisfies the requirements formula, i.e. the refinement relation between requirements and implementations is taken to be trajectory inclusion. There is a straightforward proof that undecidability of validity of formulae of the (linear-time) requirements language implies undecidability of trajectory inclusion between control devices and requirements formulae. With trajectory inclusion as refinement relation, undecidability of refinement between reactive controllers and requirements is thus a direct consequence of undecidability of the requirements language.

Although this argument is generally accepted as a rigorous proof that automatic verification and automatic synthesis of reactive control systems is impossible for sufficiently expressive temporal logics like e.g. the Duration Calculi, we feel that it is not completely stringent. While it is mathematically consistent, it builds upon the postulate that the refinement relation between requirements and implementations has to be trajectory inclusion. However, trajectory inclusion as implementation relation has been justified by the tradition of the field rather than by a thorough investigation of its physical appropriateness to embedded control. It is quite obvious that trajectory inclusion is a sufficient condition for correctness of embedded controllers, but it is not that clear whether it is also necessary. Indeed, the observation made in section 2.1 suggests that trajectory inclusion may not be a necessary condition, as it assumes that any small-scale difference between trajectories is observable. Requiring trajectory inclusion may thus be overly demanding, possibly leading to undecidability where it is not inherent to the design task.

An abbreviated version of this chapter has been published in [Frä96b].

7.1 Controller design from Duration Logic specifications

In this section, we review the argument that undecidability of a linear-time temporal logic implies undecidability of trajectory inclusion between controllers and formulae of the logic, thus providing a proof that trajectory inclusion between reactive controllers and DL formulae is undecidable. In particular, it is shown that with trajectory inclusion as satisfaction relation, undecidability of Duration Logic implies that it is in general undecidable whether an embedded controller satisfies a control problem specified in DL and that it is furthermore in general impossible to have automatic controller synthesis from DL. We start with a definition of control problems.

Definition 7.1 (Control problem). *A* control problem *is a pair (Req, I), where Req is a formula of Duration Logic specifying the admissible controller behaviours and $I \subseteq Varname$ is a set of variable names specifying the inputs to the controller, i.e. those variables that the controller cannot control and hence is not allowed to constrain in their evolution over time.*

A controller C satisfies the requirement *Req iff any possible trajectory of C — we follow the common assumption that a set of possible trajectories can be naturally assigned to controllers — satisfies Req.*

A controller C is said to solve the control problem *(Req, I) iff C has inputs I and satisfies Req.*

A key issue of control problems is that they bear a signature which distinguishes between inputs I and outputs $free(Req)\setminus I$. Most controller technologies will require such a distinction in order to maintain technology-dependent constraints that are essential to the functionality of the devices. One goal of these constraints always is to ensure that inputs can be driven by the environment, i.e. follow the environment evolution. The inputs I are thus components of the state space that are solely controlled by the environment — the controller can just observe their evolution and react upon it. Therefore, no controller can constrain its inputs at any time.[1] The outputs $free(Req) \setminus I$, on the other hand, should be controlled by the controller and may thus be constrained in their evolution by the device. Since inputs are under the sole control of the environment and cannot be constrained by the controller, the possible trajectories of a controller contain every possible input behaviour. Therefore, a controller cannot contribute to requirements that are in terms of inputs only:

Lemma 7.2. *Let ϕ be a duration formula and let C be an arbitrary controller with inputs $I \supseteq free(\phi)$. Then C satisfies ϕ iff ϕ is valid.*

Proof. It is obvious that validity of ϕ implies that C satisfies ϕ, as the trajectories of C are necessarily a subset of the universe $Traj$ of trajectories. For the converse implication observe that satisfaction of ϕ by a trajectory tr does only depend on the valuation that tr assigns to variables in $free(\phi)$. Assume that ϕ is invalid and let $tr \not\models \phi$. As C has inputs I,

[1] This is not to say that the functionality of the controller may not depend on restrictions of the possible input behaviours. But such restrictions are *assumptions* on the input behaviours occurring in the application domain. These assumptions have *to be guaranteed by the environment.* They cannot be ensured by the control device itself.

C has a trajectory tr' that coincides with tr on all variables in $I \supseteq \mathit{free}(\phi)$. As satisfaction of ϕ depends only on the valuation of its free variables, $tr' \not\models \phi$ follows. I.e., C does not satisfy ϕ. ∎

This lemma has direct consequences for the feasibility of automatic procedures in DL-based controller development.

7.1.1 Model checking

One of the immediate consequences concerns automatic verification of controllers: model checking is impossible for any class of controllers that respects above notion of input.

Theorem 7.3 (DL-based model-checking is impossible). *It is in general undecidable whether a controller solves a DL-based control problem.*

Proof. Consider a control problem (Req, I) with $I \supseteq \mathit{free}(\mathit{Req})$ and a controller C that has inputs I. As C has inputs I, C solves the control problem (Req, I) iff C satisfies Req. According to lemma 7.2, C satisfies Req iff Req is a valid formula. Hence, deciding whether C solves (Req, I) amounts to deciding validity of Req, which is in general impossible due to theorem 6.1. ∎

7.1.2 Automatic synthesis

Aiming at solutions to control problems we are also interested in synthesis methods that derive controllers from control problems. Such a synthesis method can be understood as a partial mapping *synt* from control problems to controllers. Let *synt* be a (not necessarily effective) partial function from the set of control problems to some class of controllers. We will now define when *synt* provides a sound and complete synthesis method for control problems.

Definition 7.4 (Soundness and completeness of a synthesis procedure). *We say*

1. *that synt is* sound *iff it maps control problems to solutions thereof, and*
2. *that synt is* complete *iff its domain contains all control problems that are solvable by controllers of the considered class.*

The interesting question is whether there is a sound and complete *mechanic* synthesis method for some class of controllers from DL-based control problems. Unfortunately, the answer is negative for any class of controllers providing inputs that are controlled by the environment.

Theorem 7.5 (Mechanic synthesis is impossible). *If $\mathcal{C}$ is a class of controllers that contains controllers with sufficiently large input sets then there is no effective procedure that generates solutions to each control problem that is solvable in $\mathcal{C}$. I.e., any synthesis procedure from DL-based control problems to controllers in $\mathcal{C}$ is necessarily unsound or incomplete or ineffective.*

'Sufficiently large' here means that the largest number $\mathrm{in}_{\max}$ *of inputs offered must be such that validity of DL formulae containing at most* $\mathrm{in}_{\max}$ *free variables is undecidable.* $\mathrm{in}_{\max} = 3$*, i.e. availability of controllers with three inputs, suffices.*[2]

Proof. Let *synt* be a sound and complete mapping from control problems to a class $\mathcal{C}$ of controllers offering sufficiently large input sets. Let ϕ be a DL formula for which $\mathit{free}(\phi)$ coincides with the input set of some controller $C \in \mathcal{C}$.

According to lemma 7.2, C solves the control problem $(\phi, \mathit{free}(\phi))$ iff ϕ is valid. On the other hand, again according to lemma 7.2, the control problem $(\phi, \mathit{free}(\phi))$ cannot have a solution if ϕ is invalid. Therefore, $(\phi, \mathit{free}(\phi))$ has a solution in $\mathcal{C}$ iff ϕ is valid. As *synt* is sound and complete it follows that $(\phi, \mathit{free}(\phi)) \in \mathrm{dom}\,(\mathit{synt})$ iff ϕ is valid. Thus, an effective mapping *synt* would provide a decision procedure for those formulae of Duration Logic that have sets of free variables that coincide with input sets offered by $\mathcal{C}$. As $\mathcal{C}$ offers sufficiently large input sets to render this subset of the DL formulae undecidable, this implies that *synt* cannot be effective. ∎

Theorems 7.3 and 7.5 show that undecidability of the underlying logic implies impossibility of automatic synthesis and verification methods for embedded controllers. Similar arguments have been given before for other linear-time temporal logics as well as for Duration Calculus. The latter argument has made the ProCoS II project consider only manual and user-assisted, semi-automatic controller design methods.

However, all these arguments are built upon the postulates that the refinement relation has to be trajectory inclusion. We doubt that this postulate follows from physical properties of the application domain, given that the trajectories used as models allow to encode more information in the phase-space than information-theoretically possible for any finite embedded controller, as has been shown in section 2.1. Indeed, at least the proof of undecidability of the $\{\lceil P \rceil, l = k\}$-fragment of DL crucially relies on the potential of encoding infinite information within compact Cartesian products of state and time. As theorems 7.3 and 7.5 are directly derived from those undecidability results, the observation of section 2.1 applies. Consequently, these two theorems may rely on artefacts of the formalisms rather than on inherent complexity of the application domain. Consequently, mechanic synthesis and verification procedures may exist despite theorems 7.3 and 7.5.

[2] The number 3 comes from a modification of the proof of undecidability of the $\{\lceil P \rceil, l = k\}$-fragment in [ZHS93, Theorem 11]. The original proof uses $k+3$ variables to encode a two-counter machine with k lines of program. k of those variables encode the program counter, one is a marker, and the remaining two are used for encoding the two counters. Obviously, encoding the program counter is even less demanding than encoding the other counters, as it ranges over a finite domain. Hence, the program counter can be encoded the same way that the other counters are encoded, using one discrete, two-valued variable that encodes counter value n through n oscillations within an interval of fixed length k. This reduces the number of variables to 4. Using a second marker, the three different variables that encode the two counters and the program counter can be shared. This removes two variables used in counter encoding, but adds one further marker variable, yielding a total number of 3.

7.2 Towards automatic procedures for DL-based controller design.

The proof patterns for undecidability discussed in section 2.1, which underly [ZHS93, Theorem 11], rely on encoding infinite information in compact Cartesian products of state and time, whereas only a finite part thereof can become effective in any embedded control application due to the finite capacity of sensor and actuator channels. This means that these undecidability results depend on distinguishing between some trajectories which would be causally equivalent within any embedded control application. An immediate consequence of that lacking discriminative power of the controlled environments is that much weaker implementation relations than trajectory inclusion, which assumes that any small-scale differences between trajectories are indeed observable, are sound. Accordingly weakening the implementation relation may yield a simplification of the implementation problem for controller specifications. The similarity of the underlying argument to the essential idea of *full abstractness* of denotational semantics, which has been extensively studied for sequential systems, but hardly for dense-time models of embedded or even hybrid control, is not accidental.

Analyzing the discriminative power of the components encountered in embedded control, particularly of interfaces, thus becomes a central issue. In part II of this thesis, a fairly general model of the reactivity constraints of interfaces is used for obtaining a synthesis procedure for synchronous controllers from DL. In part III, we do even apply a analysis of the transistor level of CMOS VLSI to identify sufficient reactivity constraints for gaining decidability of correctness of asynchronous VLSI with respect to specifications from the $\{\lceil P \rceil, l = k\}$-fragment of DL.

It is pleasing to see that the results are in complete accordance with the expectations that can be drawn from the analysis of physical relevance of undecidability results in chapter 2: it is only undecidability on finitely variable models which was found to lack physical interpretation, not undecidability on n-bounded models. Hence, we may expect to completely remedy the lack of mechanic verification and synthesis procedures for the $\{\lceil P \rceil, l = k\}$-fragment, which is undecidable on finitely variable but decidable on n-bounded models, but not for full DL, which is undecidable on both. These expectations coincide with our findings: while the automatic verification method for asynchronous circuits against specifications from the $\{\lceil P \rceil, l = k\}$-fragment is sound and complete, a certain incompleteness remains for the synthesis procedure dealing with full DL. However, this incompleteness is marginal and pragmatically irrelevant such that the result of our analysis can be considered a success even in that case.

Part II

Synthesis of Synchronous Controllers

While the undecidability results indicate that automatic synthesis of embedded controllers from Duration Logic is impossible according to theorem 7.5, we exploit interface latencies for setting up such a procedure and apply it to the ProCoS gas-burner case study.

8

Synthesizing Synchronous Controllers from DL Specifications

Over the past decade, compilation and automatic synthesis of hardware from abstract descriptions has gained considerable scientific interest and economic significance. A wide range of abstraction levels for the source language of such procedures has been studied, reaching from hardware-oriented structure descriptions over algorithmic descriptions by imperative languages (in a broad sense, including process calculi) to temporal logics. The obvious objective of studying more abstract source languages has been to allow the user to say *what* he or she wants the circuit to do rather than *how* to achieve this.

Although being convinced that the vocabulary of the Duration Calculus, being tailored towards a non-operational description of time-dependent behaviour, lends itself to such a task, the ProCoS consortium refrained from proposing DC as a source language for hardware synthesis — instead, they proposed an occam-dialect [HHF+94, HPB93]. The reason is that the ProCoS project, adopting trajectory inclusion as its general refinement relation, took the undecidability results concerning Duration Calculus as an indication that neither fully automatic verification methods nor automatic synthesis tools could exist for most of the design steps from Duration-Calculus-based controller specification down to the actual implementation of an embedded controller. As mentioned before, we feel that this argument is not completely stringent, because taking trajectory inclusion as implementation relation has been justified by the tradition of the field rather than by a thorough investigation of its physical appropriateness to embedded control. It is quite obvious that trajectory inclusion is a sufficient condition for correctness of embedded controllers, but it is not that clear whether it is also necessary. That means that requiring trajectory inclusion may be overly demanding, possibly leading to undecidability where it is not inherent to the design task.

In this chapter, we exploit this insight for development of an automatic synthesis procedure for synchronously clocked controllers from specifications in Duration Logic. As far as

A self-contained version of this chapter, including chapters 2 and 7 and excerpts from chapter 5, has been presented at the 'Formal Techniques in Real-Time and Fault-Tolerant Systems' 1996 conference and appeared as [Frä96b]. The material covered has furthermore been presented at the 4th ProCoS Working Group Meeting, March 1996, Oldenburg, Germany and the 'Hardware Synthesis and Verification Workshop', August 1996, Cornell University, Ithaca, USA [Frä96a].

we know, this is the first positive result concerning feasibility of automatic synthesis from dense-time Duration Calculus.

8.1 Synchronously clocked controllers

We start with an explanation of the implementation "technology" we are aiming at. As usual in the field, a somewhat simplified view of embedded control hardware is used, ignoring effects like pre-charge times of input latches and the possible consequences, particularly meta-stable behaviour. Research on resolving some of these simplifications has been performed by the author (cf. chapter 12), but within this chapter we will take advantage of the almost ubiquitous simplifications, which are by the way also part of commercial hardware description languages like VHDL [IEEE88] and Verilog HDL [SSM$^+$93]. In the following, a (synchronously) clocked controller will be a device advancing in steps of a constant length δ (the clock period), sampling its inputs at clock edges, and thereupon selecting its next state and the output valid for the next clock period. A clocked controller is thus an abstract representation of a synchronous circuit with clock period δ. For simplicity, we assume in the remainder that the clock period δ coincides with the time unit, i.e. $\delta = 1$.[1]

Definition 8.1 (Clocked controller). *A* (synchronously) clocked controller *is a 5-tuple* $C = (I, O, \Sigma, \sigma_0, T)$, *where*

- I *is a set of variable names (the input ports),*
- O, *with* $I \cap O = \emptyset$, *is a set of variable names (the output ports),*
- Σ *is a finite, nonempty set of states,*
- $\sigma_0 \in \Sigma$ *is the initial state,*
- $T \subseteq \Sigma \times \alpha \times \Sigma$, *where the alphabet* α *is* $(I \cup O) \to \mathbb{B}$ *(i.e. assigns binary values to input and output ports), is the transition relation, and*
- T *is* input-open, *which means that from any state* $\sigma \in \Sigma$ *there is a transition for each possible input* $i \in I \to \mathbb{B}$. *Formally,* T *is* input-open *iff for each* $\sigma \in \Sigma$ *and* $i \in I \to \mathbb{B}$ *there is* $\sigma' \in \Sigma$ *and* $o \in O \to \mathbb{B}$ *with* $(\sigma, i \oplus o, \sigma') \in T$.

Clocked controllers evolve over time, giving rise to runs that are related to real-time. The essential idea is that a clocked controller makes one step every time unit. The step depends on the current state and on the input values immediately before the transition. The inputs are sampled at the transition such that the step depends only on the last input value occurring before the transition and not on intermediate oscillations of inputs. As the controller makes one step every time unit, outputs and the internal state of the controller are assumed to change only at multiples of the time unit, being constant between steps.

[1] In [Frä96b] we dealt with more general clock rates. However, if $\delta \in \frac{1}{\mathbb{N}}$, i.e. if the clock rate of the devices is a multiple of the time unit used for describing the environment dynamics, the clock rate can be adjusted by accordingly adjusting the time unit. As this usually applies in hardware design we may simplify matters without loss of generality.

Definition 8.2 (Runs and trajectories of cl. controllers). *A finitely variable function $r : \mathit{Time} \to \mathit{State}$, with $\mathit{State} = I \cup O \cup \{st\} \to \mathbb{B} \cup \Sigma$ (type-consistent, i.e. I and O being mapped to $\mathbb{B}$ and $\{st\}$ being mapped to Σ), is* a run of the clocked controller $C = (I, O, \Sigma, \sigma_0, T)$ *iff*

- *$r(0)(st) = \sigma_0$, i.e. the clocked controller starts in the initial state,*
- *For each $v \in O \cup \{st\}$, the mapping $t \mapsto r(t)(v)$ is constant on $[k, k+1)$ for each $k \in \mathbb{N}$, i.e. the internal state and the output values remain constant between transitions, which happen at time instants that are multiples of the time unit,*
- *For each $k \in \mathbb{N} \setminus \{0\}$, $(\overleftarrow{\sigma}, (\overleftarrow{i} \oplus o), \sigma) \in T$, i.e. transitions are taken according to the transition relation T, where $\overleftarrow{\sigma} = r(k-1)(st)$ is the previous state, $\overleftarrow{i} = \lim_{t<k} r(t)\big|_I$ is the input just before the transition, $o = r(k)\big|_O$ is the resulting output, and $\sigma = r(k)(st)$ is the resulting state.*

  Note that the limits used in defining $\overleftarrow{i}$ do always exist due to finite variability of r.

$tr \in \mathit{Traj}$ is a trajectory of the clocked controller $C = (I, O, \Sigma, \sigma_0, T)$ *iff there is a run r of C such that $\forall t \in \mathit{Time} . \forall v \in I \cup O . r(t)(v) = tr(t)(v)$. The* set of trajectories of a clocked controller C *is denoted $\mathcal{C}[\![C]\!]$.*

Trajectories of clocked controllers are very similar to their runs, the differences being that they hide the internal state st of the clocked controller, which is considered to be externally invisible, and the slightly changed signature. Note that a clocked controller does not constrain its inputs, i.e. that there is at least one trajectory for every possible evolution of the inputs.

Based on the definition of trajectories of clocked controllers, it is straightforward to define when a clocked controller satisfies a DC formula. Following the tradition, we start with a criterion based on trajectory inclusion. The formal definition is parameterized with a semantics $[\![\cdot]\!]$ of clocked controllers, as we will later encounter a second, non-standard semantics of controllers. For the moment, the reader may read the definition as if it was only with respect to the standard semantics $\mathcal{C}[\![\cdot]\!]$ of clocked controllers.

Definition 8.3 (Satisfaction). *Let $C = (I, O, \Sigma, \sigma_0, T)$ be a clocked controller and ϕ a Duration Logic formula. We say that C* satisfies *ϕ (with respect to some semantics $[\![\cdot]\!]$ of clocked controllers), iff $[\![C]\!] \subseteq \mathcal{M}[\![\phi]\!]$, i.e. iff any trajectory of C satisfies ϕ.*

We say that C solves the control problem *$(\mathit{Req}, I_{\mathit{Req}})$ (with respect to some semantics $[\![\cdot]\!]$ of clocked controllers) iff $I \supseteq I_{\mathit{Req}}$ and $[\![C]\!] \subseteq \mathcal{M}[\![\mathit{Req}]\!]$, i.e. iff the controller respects the inputs I_{Req} and satisfies the requirements.*

The interesting question is whether there is a sound and complete *mechanic* synthesis procedure for clocked controllers from DC-based control problems. Unfortunately, the answer is negative, as far as the standard semantics of clocked controllers is concerned. As clocked controllers do not constrain their inputs at any time, and as satisfaction between controllers and requirements is trajectory inclusion, we are faced with an instance of the situation studied in section 7.1. Hence, theorem 7.5 applies, yielding

Corollary 8.4 (Impossibility of mechanic synthesis of clocked controllers). *There is no effective procedure that generates solutions with respect to the standard semantics $\mathcal{C}[\![\cdot]\!]$ for each solvable control problem.* ▟

This means that automatic synthesis from DC specifications is impossible with respect to the standard semantics of clocked controllers. Nevertheless, as theorem 7.5 is based on undecidability of DL, the observations of section 2.1 suggest that there may be computationally simpler non-standard semantics which are nevertheless operationally adequate.

8.2 A non-standard interpretation of clocked controllers

We will now start with what might at first look like a formal game hardly related to the original problem: we define a non-standard semantics to clocked controllers and show that with this semantics the synthesis problem can be mechanically solved. As this non-standard semantics is considerably simpler than the original one, it is not obvious whether it will be helpful to the original problem. However, in section 8.3.1 we will show that the two semantics are well-enough related to accept solutions with respect to the non-standard semantics also as reasonable controllers with respect to the standard semantics. The argument takes advantage of the observations concerning physical interpretation of undecidability results put forward in section 2.1.

Definition 8.5 (Non-standard semantics of clocked controllers). *The non-standard semantics $\mathcal{D}[\![\cdot]\!]$ assigning trajectory sets to clocked controllers is defined by*

$$\mathcal{D}[\![(I, O, \Sigma, \sigma_0, T)]\!] \stackrel{\text{def}}{=} \mathcal{C}[\![(\emptyset, I \cup O, \Sigma, \sigma_0, T)]\!] \ .$$

Intuitively, there are two differences between the standard semantics $\mathcal{C}[\![\cdot]\!]$ and the non-standard semantics $\mathcal{D}[\![\cdot]\!]$ of clocked controllers. First, in $\mathcal{C}[\![\cdot]\!]$ the inputs of controllers remain unconstrained, being allowed to change freely over time, whereas in $\mathcal{D}[\![\cdot]\!]$ they are constant between transitions, changing at most in time instants which are multiples of the clock period. Second, $\mathcal{C}[\![\cdot]\!]$ models sampling of inputs on clock edges such that in $\mathcal{C}[\![\cdot]\!]$ the input values *immediately before* a transition determine the output for the *following* clock period. In $\mathcal{D}[\![\cdot]\!]$, there is no such temporal asymmetry between inputs and outputs such that input values and output values they determine occur in the same time slot.

A consequence is that the non-standard interpretation of clocked controllers is closely related to finite automata on infinite words. Intuitively, the non-standard-trajectories of a clocked controller correspond to a regular ω-language [Tho90], where each letter is interpreted as a state persisting for one unit of real-time. This intuitive correspondence can be made formal through the following definition.

Definition 8.6 (Timed V-regular sets). *Let $V \subseteq$ Varname be a set of variable names. A set $T \subseteq$ Traj is called* timed V-regular *iff T is the image of a regular ω-language under* *the mapping* $timed_V : (V \to \mathbb{B})^\omega \to Traj$,

$$timed_V(\mathcal{L}) \stackrel{\text{def}}{=} \left\{ tr \in Traj \;\middle|\; \begin{array}{l} \exists\, w \in \mathcal{L}\,. \\ \quad \forall\, e \in Time, v \in V\,. \\ \qquad \Big(tr(e)(v) = w(\lfloor e \rfloor)(v)\Big) \end{array} \right\} .$$

Note that $timed_V$ leaves variables outside V unconstrained in the trajectories assigned. Concerning the variables in V, $timed_V(\mathcal{L})$ simply unrolls the words of the ω-language $\mathcal{L} \subseteq (V \to \mathbb{B})^\omega$ to real-time by making each letter persist for one time unit.

Now, the formal correspondence between clocked controllers and regular ω-languages is

Lemma 8.7 (Clocked controllers correspond to regular ω-languages). *The set $\mathcal{D}[\![C]\!]$ of non-standard-trajectories of a clocked controller C with inputs I and outputs O is timed $(I \cup O)$-regular.*

Proof. Suppose $C = (I, O, \Sigma, \sigma_0, T)$ is a clocked controller and $\texttt{fail} \notin \Sigma$.

Take the finite automaton $Aut\,(C)$ over the alphabet $I \cup O \to \mathbb{B}$ that has state set $\Sigma \uplus \{\texttt{fail}\}$, initial state σ_0, final states Σ, and the transition relation $T_{\mathcal{A}}$ defined by

$$(\sigma, a, \sigma') \in T_{\mathcal{A}} \quad \text{iff} \quad \begin{cases} (\sigma, a, \sigma') \ , \in T & \text{or} \\ \sigma' = \texttt{fail} \ \wedge \ \forall\, \tilde{\sigma} \in \Sigma\,.\,(\sigma, a, \tilde{\sigma}) \notin T \ , & \text{or} \\ \sigma = \sigma' = \texttt{fail} \ . & \end{cases}$$

Among the different ω-languages over alphabet $(V \to \mathbb{B})^\omega$ that are definable by the finite automaton $Aut\,(C)$ via different acceptance conditions, we are interested in the language $\mathcal{L}^F_{Aut(C)}$ of all those infinite words that have *all* their finite prefixes accepted by $Aut\,(C)$. Formally, for any finite automaton A over an alphabet α,

$$\mathcal{L}^F_A \stackrel{\text{def}}{=} \left\{ w \in \alpha^\omega \,\middle|\, \forall\, v \in \alpha^*\,.\, \Big(v \leq w \Rightarrow v \in \mathcal{L}_A\Big) \right\} ,$$

where $\leq$ is the prefix relation and $\mathcal{L}_A$ is the set of finite words accepted by A.

It is straightforward to check from the definitions that $tr \in \mathcal{D}[\![C]\!]$ iff there is $w \in \mathcal{L}^F_{Aut(C)}$ with $tr \in timed_{I \cup O}(\{w\})$. I.e. $\mathcal{D}[\![C]\!] = timed_{I \cup O}(\mathcal{L}^F_{Aut(C)})$. ∎

Indeed, as can be seen from the proof, the non-standard semantics of a clocked controller even corresponds to a regular language in the class F of the Borel hierarchy [Tho90], which is a considerably simpler class than general regular ω-languages.

In the following, we exploit the correspondence between the non-standard semantics of clocked controllers and regular ω-languages expressed in lemma 8.7 for reducing the synthesis problem to a problem of regular games. In order to do so, a reformulation of the satisfaction problem between controllers and DL formulae in terms of timed V-regular languages is used.

Lemma 8.8. *A clocked controller $C = (I, O, \Sigma, \sigma_0, T)$ satisfies a requirement ϕ with respect to the non-standard semantics iff $\mathcal{D}[\![C]\!]$ is contained in the largest timed $(I \cup O)$-regular set that is contained in $\mathcal{M}[\![\phi]\!]$.*

Proof. By lemma 8.7, $\mathcal{D}[\![C]\!]$ is itself timed $(I \cup O)$-regular. Hence, $\mathcal{D}[\![C]\!] \subseteq \mathcal{M}[\![\phi]\!]$, i.e. C satisfies ϕ with respect to the non-standard semantics, iff $\mathcal{D}[\![C]\!]$ is contained in the largest timed $(I \cup O)$-regular set that is contained in $\mathcal{M}[\![\phi]\!]$. ▟

Furthermore, the largest timed V-regular set that is contained in $\mathcal{M}[\![\phi]\!]$ has an effective construction if $V \supseteq \mathit{free}(\phi)$.

Lemma 8.9. *Let $V \subseteq \mathit{Varname}$ and suppose ϕ is a DL formula with $\mathit{free}(\phi) \subseteq V$.*
The trajectory set $\mathcal{T} \stackrel{\text{def}}{=} \mathit{timed}_V(\mathcal{L})$, where

$$\mathcal{L} \stackrel{\text{def}}{=} \left\{ w \in (V \to \mathbb{B})^\omega \;\middle|\; \begin{array}{l} \forall v \in (V \to \mathbb{B})^* . \\ \quad \left(v \leq w \Rightarrow v \notin \bigcup_{r \in [0,1)} \mathcal{L}_{0,r}(\neg\phi)\right) \end{array} \right\} \;,$$

is the largest timed V-regular language contained in $\mathcal{M}[\![\phi]\!]$. A finite automaton A_ϕ with $\mathcal{L}^F_{A_\phi} = \mathcal{L}$ can be effectively constructed.

Proof. According to corollary 6.21 there is an effective construction for a finite automaton accepting $\bigcup_{r \in [0,1)} \mathcal{L}_{0,r}(\neg\phi)$. As negation of regular languages is effective, an automaton A_ϕ accepting $(V \to \mathbb{B})^* \setminus \bigcup_{r \in [0,1)} \mathcal{L}_{0,r}(\neg\phi)$ can be effectively constructed. By definition of $\mathcal{L}^F$,

$$\mathcal{L}^F_{A_\phi} = \left\{ w \in (V \to \mathbb{B})^\omega \;\middle|\; \begin{array}{l} \forall v \in (V \to \mathbb{B})^* . \\ \quad \left(v \leq w \Rightarrow v \notin \bigcup_{r \in [0,1)} \mathcal{L}_{0,r}(\neg\phi)\right) \end{array} \right\} = \mathcal{L} \;.$$

This shows

1. that A_ϕ with $\mathcal{L}^F_{A_\phi} = \mathcal{L}$ can be effectively constructed and
2. that $\mathcal{T}$ is timed V-regular, as $\mathcal{T} = \mathit{timed}_V(\mathcal{L}) = \mathit{timed}_V(\mathcal{L}^F_{A_\phi})$ and $\mathcal{L}^F_{A_\phi}$ is a regular ω-language.

It remains to be shown that $\mathcal{T}$ is the largest timed V-regular set contained in $\mathcal{M}[\![\phi]\!]$. Therefore let $\mathcal{T}' \subseteq \mathit{Traj}$ be an arbitrary timed V-regular set that is contained in $\mathcal{M}[\![\phi]\!]$. We will show that $\mathcal{T}' \subseteq \mathcal{T}$.

As $\mathcal{T}'$ is timed V-regular there is a regular ω-language $\mathcal{L}' \subseteq (V \to \mathbb{B})^\omega$ with $\mathcal{T}' = \mathit{timed}_V(\mathcal{L}')$. Let $w \in \mathcal{L}'$. By definition of timed_V there is $\mathit{tr} \in \mathit{timed}_V(\mathcal{L}') \cap \mathit{DTraj}$. As $\mathit{DTraj} = \mathrm{dom}\,(\mathit{wrd}_V)$, the digitization $\mathit{wrd}_V(\mathit{tr})$ is defined.[2] By definition of wrd_V and timed_V, $\mathit{wrd}_V(\mathit{tr}) = w$ as $\mathit{tr} \in \mathit{timed}_V(\{w\})$. As $\mathit{timed}_V(\{w\}) \subseteq \mathit{timed}_V(\mathcal{L}') \subseteq \mathcal{M}[\![\phi]\!]$ we have $\mathit{tr} \models \phi$. I.e., $\mathit{tr}, [0,e] \models \phi$ for each $e \in \mathit{Time}$. Consequently, $\mathit{tr}', [0,e] \not\models \neg\phi$ for each $e \in \mathit{Time}$ and each $\mathit{tr}' \in \mathit{DTraj}$ with $\mathit{wrd}_V(\mathit{tr}')[0, \lfloor e \rfloor] = \mathit{wrd}_V(\mathit{tr})[0, \lfloor e \rfloor] = w[0, \lfloor e \rfloor]$.

Hence, for each $e \in \mathit{Time}$,

$$w[0, \lfloor e \rfloor] \quad \notin \quad \left\{ \mathit{wrd}_V(\mathit{tr}')[0, \lfloor e \rfloor] \;\middle|\; \begin{array}{l} \mathit{tr}' \in \mathit{DTraj}, e \in \mathit{Time}, \\ \mathit{tr}', [0,e] \models \neg\phi \end{array} \right\}$$

[2] Cf. page 42 for the definition of wrd_V.

$$= \bigcup_{r\in[0,1)} \left\{ wrd_V(tr')[0,\lfloor e\rfloor] \;\middle|\; \begin{array}{l} tr' \in DTraj, e \in Time, \\ e = \lfloor e\rfloor + r, \\ tr', [0,e] \models \neg\phi \end{array} \right\}$$
$$= \bigcup_{r\in[0,1)} \mathcal{L}_{0,r}(\neg\phi) \ .$$

But then

$$w \in \left\{ w \in (V \to \mathbb{B})^\omega \;\middle|\; \begin{array}{l} \forall v \in (V \to \mathbb{B})^* . \\ \quad \left(v \leq w \Rightarrow v \notin \bigcup_{r\in[0,1)} \mathcal{L}_{0,r}(\neg\phi)\right) \end{array} \right\} = \mathcal{L} \ .$$

As $w \in \mathcal{L}'$ was arbitrarily chosen this shows that $\mathcal{L}' \subseteq \mathcal{L}$, which implies $timed_V(\mathcal{L}') \subseteq timed_V(\mathcal{L})$. Consequently, $\mathcal{T}' \subseteq \mathcal{T}$ because $timed_V(\mathcal{L}') = \mathcal{T}'$ and $timed_V(\mathcal{L}) = \mathcal{T}$. As $\mathcal{T}'$ was arbitrarily chosen among the timed V-regular sets that are contained in $\mathcal{M}[\![\phi]\!]$ this shows that $\mathcal{T}$ is the largest timed V-regular set that is contained in $\mathcal{M}[\![\phi]\!]$. ∎

An immediate consequence of lemmata 8.8 and 8.9 is that the controller correctness problem has an effective reduction to a problem of regular ω-languages.

Corollary 8.10. *C satisfies ϕ with respect to the non-standard semantics, i.e. $\mathcal{D}[\![C]\!] \subseteq \mathcal{M}[\![\phi]\!]$, iff $\mathcal{L}^F_{Aut(C)} \subseteq \mathcal{L}^F_{A_\phi}$.*

Proof. According to lemma 8.8, C satisfies ϕ if and only if $\mathcal{D}[\![C]\!]$ is contained in the largest timed $(I \cup O)$-regular set that is contained in $\mathcal{M}[\![\phi]\!]$. By lemma 8.9, the largest $(I \cup O)$-regular set that is contained in $\mathcal{M}[\![\phi]\!]$ is $timed_{I\cup O}(\mathcal{L}^F_{A_\phi})$. Hence, C satisfies ϕ iff $\mathcal{D}[\![C]\!] \subseteq timed_{I\cup O}(\mathcal{L}^F_{A_\phi})$. As $\mathcal{D}[\![C]\!] = timed_{I\cup O}(\mathcal{L}^F_{Aut(C)})$, it remains to be shown that $timed_{I\cup O}(\mathcal{L}^F_{Aut(C)}) \subseteq timed_{I\cup O}(\mathcal{L}^F_{A_\phi})$ iff $\mathcal{L}^F_{Aut(C)} \subseteq \mathcal{L}^F_{A_\phi}$. This holds, as the definition of $timed_V$ immediately implies that $timed_V(X) \subseteq timed_V(Y)$ iff $X \subseteq Y$. ∎

Note that $\mathcal{L}^F_{Aut(C)} \subseteq \mathcal{L}^F_{A_\phi}$ is a criterion involving just regular ω-languages that can be effectively constructed from the original correctness problem $\mathcal{D}[\![C]\!] \subseteq \mathcal{M}[\![\phi]\!]$ due to effectiveness of all the automata constructions involved. We will exploit this for an effective reduction of the controller design problem to a regular infinite game [Tho90].

As our controllers operate on sets of Boolean-valued variables — their ports — we give here a specialized description of infinite games that is tailored towards this special class of alphabets. Here, an *infinite game* is a triple (I, O, Γ), where I and O are disjoint sets of variable names, and $\Gamma \subseteq (I \cup O \to \mathbb{B})^\omega$ is the success criterion for the controller. The environment takes moves in $I \to \mathbb{B}$ and the controller takes moves in $O \to \mathbb{B}$. Both do their moves in turn. The controller wins iff the resulting infinite sequence $(i_j \oplus o_j)_{j\in\mathbb{N}}$ of move pairs, where $\oplus$ denotes the union of functions with disjoint domain, is in Γ. If Γ is a regular ω-language then (I, O, Γ) is called a *regular infinite game*.

An interesting question on infinite games is whether one of the players has a finite-state winning strategy. Intuitively, a *finite-state winning strategy* is a finite automaton that tells the player what to do next, thereby guaranteeing a sure win. E.g. a finite-state winning strategy for the controller is a finite automaton A satisfying

1. every environment move is accepted at any time, i.e. $\varepsilon \in \mathcal{L}_A$ and for each $w \in \mathcal{L}_A$ and each $i : I \to \mathbb{B}$ there is $o : O \to \mathbb{B}$ with $w \cdot (i \oplus o) \in \mathcal{L}_A$, where $\mathcal{L}_A \subseteq (I \cup O \to \mathbb{B})^*$ is the regular language accepted by A, and

2. A leads the controller to a sure win, i.e. $\mathcal{L}_A^F \subseteq \Gamma$.

A finite-state winning strategy for the environment is defined similarly, but for obvious reasons we are not interested in those here.

A classical theorem of Büchi and Landweber states that strategies in regular infinite games can be constructed effectively.

Theorem 8.11 (Reg. games are determined [BL69]). *In any regular game one of the players has a finite-state winning strategy. It can be decided effectively who has, and the finite-state winning strategy can be constructed effectively.*

This result becomes helpful to synthesis of clocked controllers through the following fact.

Lemma 8.12. *Let C be a clocked controller with inputs I and outputs O, let ϕ be a DC formula with $free(\phi) \subseteq I \cup O$. Then $\mathcal{D}[\![C]\!] \subseteq \mathcal{M}[\![\phi]\!]$, i.e. C satisfies ϕ with respect to the non-standard semantics, iff $Aut\,(C)$ is a winning strategy for the controller in the regular game $\left(I, O, \mathcal{L}_{A_\phi}^F\right)$.*

Proof. If C is a clocked controller with input set I then its transition relation is input-open with respect to I, which, by construction of $Aut\,(C)$, ensures that the first criterion for $Aut\,(C)$ being a winning strategy does always hold. Thus, $Aut\,(C)$ is a winning strategy for the controller in the regular game $\left(I, O, \mathcal{L}_{A_\phi}^F\right)$ iff the second criterion is met, i.e. iff $\mathcal{L}_{Aut(C)}^F \subseteq \mathcal{L}_{A_\phi}^F$. By corollary 8.10, $\mathcal{L}_{Aut(C)}^F \subseteq \mathcal{L}_{A_\phi}^F$ iff $\mathcal{D}[\![C]\!] \subseteq \mathcal{M}[\![\phi]\!]$. ∎

An immediate consequence of this lemma and the theorem of Büchi and Landweber is that the synthesis problem for clocked controllers can be solved under the non-standard interpretation.

Theorem 8.13 (Feasibility of automatic synthesis). *There is an effective, sound, and complete synthesis method for clocked controllers from DL-based control problems with respect to the non-standard semantics.*

Proof. By theorem 8.11 there is an effective partial function *synt* that assigns to each pair (I, O) of disjoint variable sets and each finite automaton A over the alphabet $I \cup O \to \mathbb{B}$ a finite-state winning strategy $synt(I, O, A)$ for the controller in the regular game $(I, O, \mathcal{L}_A^F)$ if such a winning strategy exists, and is undefined otherwise. As any finite-state strategy is equivalent to one with only one non-final state, *synt* yields w.l.o.g. finite-state strategies that are in the image of the mapping $Aut\,(\cdot)$ of clocked controllers to finite automata.

A partial function $\widetilde{synt}$ from control problems to clocked controllers is then defined by

$$\widetilde{synt}(\phi, I) \stackrel{\text{def}}{=} Aut^{-1}\,(synt\,(I, free(\phi) \setminus I, A_\phi)) \quad .$$

$\widetilde{synt}$ is effective by effectiveness of the automata mappings involved and of *synt*. Furthermore, soundness and completeness of $\widetilde{synt}$ follows from the corresponding properties of *synt* by lemma 8.12. ∎

As the proof shows, the regular games to be solved in DL-based controller synthesis are of the special form $(I, O, \mathcal{L}_A^F)$ for some finite automaton A, i.e. the winning conditions belong to class F of the Borel hierarchy, also called π_1^0 winning conditions [Tho95]. Such games, sometimes called safety games, can be solved by so-called *no-memory strategies* (as can be some games with more general winning conditions, see [Tho95] for an overview). The term 'no-memory strategy' refers to the fact that a winning strategy can be built on the state set of the automaton A that defines the regular game by just narrowing the transition relation. I.e., a no-memory strategy needs no additional memory beyond that provided by the state set of A. If we pictorially think of the automaton A as the game graph showing the possible moves in the game then the satisfying 'no-memory' controller is simply a fragment of A.

Therefore, a control automaton can be derived from a DL-based control problem by selecting an appropriate subset of the transitions of the case automaton A_ϕ. The following algorithm uses this technique for deciding whether a winning strategy for the controller exists and, if so, for generating it.

Algorithm 8.14 (Synthesis of clocked controllers from DL control problems). Let (ϕ, I) be a control problem. Existence of a solving clocked controller with respect to the non-standard semantics can be decided and, if applicable, such a controller can be constructed by the following algorithm.

1. **Construct the case automaton A_ϕ.**

   This can be done using the construction of section 6.3 to obtain a finite automaton accepting $\bigcup_{r\in[0,1)} \mathcal{L}_{0,r}(\phi)$ and then using complementation to build an automaton that accepts the language $(\mathit{free}(\phi) \to \mathbb{B})^* \setminus \bigcup_{r\in[0,1)} \mathcal{L}_{0,r}(\phi)$.

   The latter automaton is the desired automaton A_ϕ, as can be seen from the proof of lemma 8.9.

2. **Mark the uncontrollable states in A_ϕ.**

   According to theorem 8.13 we have to construct a winning strategy for the controller in the π_1^0-game $(I, \mathit{free}(\phi) \setminus I, \mathcal{L}_{A_\phi}^F)$. As π_1^0-games can be solved by no-memory strategies, a winning strategy can be determined by solely marking the individual states of A_ϕ as either being controllable or uncontrollable. A *state is controllable* iff the environment cannot force a visit to a non-accepting state (which would indicate that the controller looses the game) from it. Otherwise, the *state is uncontrollable*, as the environment can enforce a win of the game.

   Assume that the case automaton A_ϕ has state set Σ, initial state σ_0, final states F, and transition relation T. The uncontrollable states in Σ can be determined by computing, for $j \in \mathbb{N}$, the sets U_j of states from which the environment can enforce a visit to a non-accepting state in at most j steps. Obviously, $U_0 = \Sigma \setminus F$ and

$$U_{j+1} = U_j \cup \left\{ \sigma \in \Sigma \;\middle|\; \begin{array}{l} \exists\, i \in I \to \mathbb{B} \,. \\ \quad \forall\, o \in (\mathit{free}(\phi) \setminus I) \to \mathbb{B} \,. \\ \qquad \exists\, \sigma' \in U_j \,.\, (\sigma, (i \oplus o), \sigma') \in T \end{array} \right\} .$$

The set U of uncontrollable states, i.e. states from which the environment can force a visit to a non-accepting state in an arbitrary number of steps, is the union $\bigcup_{j \in \mathbb{N}} U_j$. Since the state set Σ is finite , the increasing sequence $(U_j)_{j \in \mathbb{N}}$ eventually becomes stable. Hence, the set $U = \bigcup_{j \in \mathbb{N}} U_j$ of uncontrollable states can be computed effectively.

3. **Determine whether the control problem (ϕ, I) has a solution.**

   Obviously, the system is uncontrollable iff the environment can force a visit to a non-accepting state from the initial state σ_0. Hence, it suffices to check whether $\sigma_0 \in U$ in order to decide whether the control problem (ϕ, I) has a solution.

   If $\sigma_0 \in U$ then the synthesis procedure is aborted as the control problem has no solution. Otherwise, the algorithm proceeds with step 4, generating a clocked controller that solves the control problem.

4. **Generate the clocked controller.**

   A win for the controller in the game $(I, \mathit{free}(\phi) \setminus I, \mathcal{L}^F_{A_\phi})$ can be achieved by always taking transitions from controllable to controllable states. Hence, the automaton C_ϕ that has the same state set Σ, the same initial state σ_0, and the same transition relation T as A_ϕ, but as final states the controllable states $\Sigma \setminus U$, is a winning strategy for the regular game $(I, \mathit{free}(\phi) \setminus I, \mathcal{L}^F_{A_\phi})$.

   Since the uncontrollable states form a sink by construction (i.e., all transitions from uncontrollable states lead to uncontrollable states), the uncontrollable states can be identified without changing the accepted language. This yields an automaton $\tilde{C}_\phi$ that has just one non-final state and is thus in the image of the mapping $Aut(\cdot)$. The preimage $Aut^{-1}\left(\tilde{C}_\phi\right)$ then provides the desired clocked controller according to theorem 8.13.

•

As theorem 8.13 shows, fully automatic synthesis from DC-based control problems is possible, albeit only with respect to the non-standard semantics of clocked controllers. On the other hand, the intended semantics of clocked controllers is clearly the standard one, not the somewhat artificial non-standard semantics. Hence, it is not at all clear whether theorem 8.13 is of any significance to the problem of designing clocked controllers that meet requirements that are specified through DL formulae. We will now expand on the observations made in section 2.1 in order to show that the theorem does indeed give a synthesis method that can cope with the original problem.

8.3 A weaker refinement criterion than trajectory inclusion

Following the observations made in section 2.1 concerning the questionable physical relevance of the undecidability proof for DL, we will now do a deeper investigation of the architecture depicted in figure 2.1 on page 12 and the engineering processes involved. It is through this investigation of the demands posed by the architecture that we will find a sufficient criterion for implementation correctness that is weaker than trajectory inclusion. This weaker notion will then be instrumental to finding a handle towards automation of controller design from DL formulae even with respect to the standard semantics of clocked controllers.

Glancing at figure 2.1 on page 12 we find that the embedded controller is in general embedded into an environment consisting of sensors, actuators, and a plant to be controlled. All these components are designed in individual development steps that we will look at in the next subsections. Doing so we locate our design activity — that of refining a specification of intended real-time control behaviour towards actual control hardware — within a waterfall model of system development. The steps of the waterfall model we will be concerned with are *safety analysis*, *sensor and actuator design*, and *controller specification*.

Safety analysis. During safety analysis, safety engineers analyze the plant, uncovering safety and liveness requirements S that the plant-controller-combination has to satisfy. I.e., safety analysis imposes a requirement

$$(\mathit{Plant} \| \mathit{Sensors} \| \mathit{Controller} \| \mathit{Actuators}) \ \mathtt{sat}\ S \ ,$$

where $\|$ is an appropriate form of parallel composition (cf. figure 2.1 on page 12) and `sat` the (perhaps informal) satisfaction relation between systems and requirements, on the further design activity. Following the lines of ProCoS, we will assume in the remainder that parallel composition is adequately modelled by intersection of the trajectory sets of the components and that satisfaction coincides with trajectory inclusion at this global level.

As safety analysis is concerned with ensuring an acceptable functionality of the plant, it is natural to assume that S contains only plant variables, i.e. does not mention internals of the controller yet to be designed.

Sensor and actuator design. In a second set of design steps, interface hardware for interfacing the embedded controller to the plant has to be developed by electrical and/or domain-specific engineers. Within a proper design activity, the engineers have to provide descriptions `sens` (respectively `act`) of the dynamic behaviour of the designed sensors (actuators, respectively), describing their sets of possible trajectories. Physically reasonable descriptions will involve a notion of *sensor and actuator latency.* A sensor is said to have *latency* t, with $t > 0$, iff it reliably reports any external stimulus that persists for at least t time units, whereas shorter stimuli are not necessarily detected (the definition for actuators is analogous). A formal description of latency that has proved physically adequate in a low-level investigation of switching hardware to be reported in chapter 12 is shown in table 8.1.

A sensor with latency t reporting a property ϕ on a Boolean variable x is described by the DL formula

$$\mathtt{sens}_t(\phi, x) \stackrel{\text{def}}{=} \lceil \phi \rceil^t \rightsquigarrow \lceil x \rceil \ \wedge \ \lceil \neg\phi \rceil^t \rightsquigarrow \lceil \neg x \rceil \ ,$$

where, according to the abbreviations introduced in chapter 5, $\lceil \phi \rceil^t$ means that the state property ϕ holds (almost) continuously for t time units, and $\Phi \rightsquigarrow \lceil \psi \rceil$ means that every occurrence of Φ in time is followed by a period of positive duration where the state formula ψ holds (almost) everywhere. Thus, $\mathtt{sens}_t(\phi, x)$ expresses that phases where property ϕ (or $\neg\phi$) persists for at least t time units will be reliably reported by x ($\neg x$, respectively).

Table 8.1: A sensor with latency t.

Another outcome of the interface design activity is the *signature* for the subsequent embedded-controller design activity. Sensor and actuator design defines an interface to the controller, which classifies some of the interface ports as inputs to the controller and others as outputs from the controller. It is thus sensor and actuator design which yields the port sets that we pay attention to during controller design.

Control requirements specification. After safety analysis and interface design, the requirements on the controller behaviour can be deduced. The result should be a *control requirement formula Req* in a suitable real-time formalism (here: Duration Logic) such that plant dynamics, sensor and actuator behaviour, and the control requirement *Req* together satisfy the safety requirements S of the safety analysis. I.e.,

$$(\mathit{Plant} \cap \mathtt{sens} \cap \mathcal{M}[\![\mathit{Req}]\!] \cap \mathtt{act}) \subseteq S \ . \tag{8.1}$$

As *Req* expresses constraints on the controller behaviour, it should be in terms of port values (and possibly auxiliary variables) only — *Req* should not resort to plant variables. Thus, $\mathit{free}(\mathit{Req}) \cap \mathit{free}(S) = \emptyset$.

But this implies that it is sufficient to demand from a clocked controller $C = (I, O, \Sigma, \sigma_0, T)$ that is to soundly control the system that it satisfies

1. $I \supseteq I_{\text{sens}}$, where I_{sens} is the set of inputs identified during sensor design, and
2. $(\mathtt{sens} \cap \mathcal{C}[\![C]\!] \cap \mathtt{act}) \subseteq_{\text{ext}} (\mathtt{sens} \cap \mathcal{M}[\![\mathit{Req}]\!] \cap \mathtt{act})$, where $\subseteq_{\text{ext}}$ denotes trajectory inclusion after projection to the externally visible variables, i.e. to the plant variables.

   Formally, $\mathit{Tr} \subseteq_{\text{ext}} \mathit{Tr}'$ for trajectory sets $\mathit{Tr}, \mathit{Tr}' \subseteq \mathit{Traj}$ iff

   $$\{t \mapsto \mathit{tr}(t)\big|_{\mathit{PlantVar}} \mid \mathit{tr} \in \mathit{Tr}\} \subseteq \{t \mapsto \mathit{tr}(t)\big|_{\mathit{PlantVar}} \mid \mathit{tr} \in \mathit{Tr}'\} \ ,$$

   where *PlantVar* is the set of plant variables.

Note that, due to the latencies that sensors and actuators impose, these conditions are considerably weaker than those covering refinement by trajectory inclusion, where the second condition was $\mathcal{C}[\![C]\!] \subseteq \mathcal{M}[\![\mathit{Req}]\!]$.

Perhaps surprisingly, criterion 2 is so much weaker than the original one that it can — although itself expressed in terms of the standard semantics — also be expressed in terms of the non-standard semantics.

8.3.1 The weaker criterion in terms of the non-standard semantics

As powerful methods have been developed for the non-standard semantics $\mathcal{D}[\![\cdot]\!]$ of controllers in section 8.2 it is worthwhile to express the new correctness criterion in terms of that semantics. Indeed, $\mathcal{C}[\![\cdot]\!]$ and $\mathcal{D}[\![\cdot]\!]$ are related in a way that fits well into the pattern of the weakened correctness criterion.

Theorem 8.15 (Correspondence of standard and non-standard semantics). *If C is a clocked controller and dynamics of the individual sensors is described by formulae of the shape shown in table 8.1 then* $\mathtt{sens} \cap \mathcal{C}[\![C]\!] \cap \mathtt{act} =_{\mathrm{ext}} \mathtt{sens}_{+1} \cap \mathcal{D}[\![C]\!] \cap \mathtt{act}$, *where* $\mathtt{sens}_{+1}$ *is the sensor behaviour with the latency of each sensor enlarged by one time unit.*

Proof. Let C be a clocked controller with inputs I. Let $d : \mathit{Traj} \to \mathit{Traj}$ be the mapping defined by

$$d(tr)(t)(v) \stackrel{\mathrm{def}}{=} \begin{cases} tr(t)(v) & \text{iff} \quad v \notin I \ , \\ \lim_{t' < \lfloor t \rfloor} tr(t')(v) & \text{iff} \quad v \in I \ . \end{cases}$$

d moves input values in time according to the different interpretation of inputs in the two controller semantics. If i is an input which has value x in trajectory tr just before a transition than i has that same value x in $d(tr)$ for a full time unit following the transition.

From the definitions it can be checked that

1. if $tr \in \mathcal{C}[\![C]\!]$ then $d(tr) \in \mathcal{D}[\![C]\!]$,
2. if $tr \in \mathcal{D}[\![C]\!]$ then each $tr' \in d^{-1}(tr)$ satisfies $tr' \in \mathcal{C}[\![C]\!]$,
3. if $tr \in \mathtt{sens}$ then $d(tr) \in \mathtt{sens}_{+1}$,
4. if $tr \in \mathtt{sens}_{+1}$ and $tr \in \mathrm{im}\,(d)$ then there is $tr' \in d^{-1}(tr)$ such that $tr' \in \mathtt{sens}$,
5. $tr \in \mathtt{act}$ iff $d(tr) \in \mathtt{act}$,
6. if $tr \in \mathcal{D}[\![C]\!]$ then $tr \in \mathrm{im}\,(d)$,
7. for each $tr \in \mathit{Traj}$, tr and $d(tr)$ are externally equivalent, where *externally equivalent* means equal after projection to the externally observable variables, i.e.

$$t \mapsto tr(t)\big|_{\mathit{PlantVar}} = t \mapsto d(tr)(t)\big|_{\mathit{PlantVar}} \ .$$

Now let $tr \in \mathtt{sens} \cap \mathcal{C}[\![C]\!] \cap \mathtt{act}$. By 1, 3, and 5, $d(tr) \in \mathtt{sens}_{+1} \cap \mathcal{D}[\![C]\!] \cap \mathtt{act}$. By 7 this shows that $(\mathtt{sens} \cap \mathcal{C}[\![C]\!] \cap \mathtt{act}) \subseteq_{\mathrm{ext}} (\mathtt{sens}_{+1} \cap \mathcal{D}[\![C]\!] \cap \mathtt{act})$.

On the other hand, let $tr \in \mathtt{sens}_{+1} \cap \mathcal{D}[\![C]\!] \cap \mathtt{act})$. By 2, 4, 5, and 6 there is a trajectory $tr' \in d^{-1}(tr)$ with $tr' \in \mathtt{sens} \cap \mathcal{C}[\![C]\!] \cap \mathtt{act}$. By property 7 this shows the remaining inclusion $(\mathtt{sens}_{+1} \cap \mathcal{D}[\![C]\!] \cap \mathtt{act}) \subseteq_{\mathrm{ext}} (\mathtt{sens} \cap \mathcal{C}[\![C]\!] \cap \mathtt{act})$. ∎

The connection between standard and non-standard semantics of clocked controllers expressed in above theorem implies

Corollary 8.16. $$\begin{array}{rcll} \mathtt{sens} \cap \mathcal{C}[\![C]\!] \cap \mathtt{act} & \subseteq_{\mathrm{ext}} & \mathtt{sens}_{+1} \cap \mathcal{M}[\![Req]\!] \cap \mathtt{act} & \textit{iff} \\ \mathtt{sens}_{+1} \cap \mathcal{D}[\![C]\!] \cap \mathtt{act} & \subseteq_{\mathrm{ext}} & \mathtt{sens}_{+1} \cap \mathcal{M}[\![Req]\!] \cap \mathtt{act} & \end{array}$$ ■

Consequently, correctness of the clocked controller C can be related to trajectory inclusion with respect to the non-standard semantics:

Corollary 8.17. *$\mathcal{D}[\![C]\!] \subseteq \mathcal{M}[\![Req]\!]$, i.e. satisfaction of ϕ by C with respect to the non-standard semantics, implies $\mathtt{sens} \cap \mathcal{C}[\![C]\!] \cap \mathtt{act} \subseteq_{\mathrm{ext}} \mathtt{sens}_{+1} \cap \mathcal{M}[\![Req]\!] \cap \mathtt{act}$.* ■

This makes $\mathcal{D}[\![C]\!] \subseteq \mathcal{M}[\![Req]\!]$ an almost, i.e. except for the slightly changed timing parameters of the sensors, sufficient condition for correctness of the clocked controller C. In order to compensate for the slight changes in the timing parameters we may ask the control engineer to deliver a slightly stronger control specification *Req* which satisfies

$$(Plant \cap \mathtt{sens}_{+1} \cap \mathcal{M}[\![Req]\!] \cap \mathtt{act}) \subseteq S \tag{8.2}$$

instead of inequality (8.1). Then we have

Theorem 8.18 (Soundness of synthesis wrt. the standard semantics). *If inequality (8.2) holds and the synthesis procedure of theorem 8.13, applied to the control problem (Req, I), yields a clocked controller C, then $(Plant \cap \mathtt{sens} \cap \mathcal{C}[\![C]\!] \cap \mathtt{act}) \subseteq S$.*

Thus, methods for solving regular games induce a sound synthesis procedure for clocked controllers from DL-based specifications.

Proof. By theorem 8.13, $\mathcal{D}[\![C]\!] \subseteq \mathcal{M}[\![Req]\!]$ holds for the generated controller C. By corollary 8.17 this implies $\mathtt{sens} \cap \mathcal{C}[\![C]\!] \cap \mathtt{act} \subseteq_{\mathrm{ext}} \mathtt{sens}_{+1} \cap \mathcal{M}[\![Req]\!] \cap \mathtt{act}$, which by (8.2) implies $(Plant \cap \mathtt{sens} \cap \mathcal{C}[\![C]\!] \cap \mathtt{act}) \subseteq S$. ■

Finally, one might ask whether there is a price to pay for gaining an effective synthesis procedure. At first glance there seems to be one, namely that the timing constraints on the real-time specification *Req* are tighter than in the original setting. Originally, *Req* was required to satisfy $Plant \cap \mathtt{sens} \cap \mathcal{M}[\![Req]\!] \cap \mathtt{act} \subseteq S$, now it is required to satisfy even $Plant \cap \mathtt{sens}_{+1} \cap \mathcal{M}[\![Req]\!] \cap \mathtt{act} \subseteq S$. As $\mathtt{sens}_{+1}$ is a larger trajectory set than $\mathtt{sens}$, there are in general DC formulae *Req* that satisfy the former, but not the latter. Hence, we might have lost completeness with our new method. *Pragmatically*, we have not, as the sensor latencies are in general some orders of magnitude larger than the clock periods of synchronous hardware, such that control specifications failing because of just one extra clock cycle of sensor latency are somewhat pathological. But even *formally*, we have not lost anything, as the next theorem shows.

Theorem 8.19 (Completeness). *If a clocked controller exists that solves the control problem (Req, I) with respect to the standard semantics then there is a specification Req' with $Plant \cap \mathtt{sens}_{+1} \cap Req' \cap \mathtt{act} \subseteq S$ such that the synthesis method succeeds for the control problem (Req', I).*

Proof. Let (Req, I) be a solvable control problem and $C = (I, O, \Sigma, \sigma_0, T)$ a solution thereof.

It is straightforward to construct a DL formula $\phi_{\mathcal{D}[\![C]\!]}$ with free variables $I \cup O$ plus $\lceil \log_2 |\Sigma| \rceil + 1$ auxiliary variables that simulates $\mathcal{D}[\![C]\!]$. The auxiliary variables are for encoding the state space Σ and for constructing a clock of period 1. The simulation is such that the set of models of $\phi_{\mathcal{D}[\![C]\!]}$ is — except for the additional constraints imposed on the auxiliary variables — identical to $\mathcal{D}[\![C]\!]$.[3] If we select the auxiliary variables such that they are not externally visible, it is obvious that $\mathtt{sens}_{+1} \cap \mathcal{D}[\![C]\!] \cap \mathtt{act} =_{\text{ext}} \mathtt{sens}_{+1} \cap \mathcal{M}[\![\phi_{\mathcal{D}[\![C]\!]}]\!] \cap \mathtt{act}$. As $\mathtt{sens} \cap \mathcal{C}[\![C]\!] \cap \mathtt{act} =_{\text{ext}} \mathtt{sens}_{+1} \cap \mathcal{D}[\![C]\!] \cap \mathtt{act}$ and as C is a solution to the control problem (Req, I) it follows immediately that $Plant \cap \mathtt{sens}_{+1} \cap \mathcal{M}[\![\phi_{\mathcal{D}[\![C]\!]}]\!] \cap \mathtt{act} \subseteq S$. Furthermore, as $\phi_{\mathcal{D}[\![C]\!]}$ itself describes the dynamics of a clocked controller with inputs I, it is obvious that the control problem $(\phi_{\mathcal{D}[\![C]\!]}, I)$ is solvable. According to theorem 8.13, the synthesis procedure will thus succeed when applied to the control problem $(\phi_{\mathcal{D}[\![C]\!]}, I)$. ∎

This result shows that one can always find a starting point for synthesis if the control problem is solvable. Nevertheless, the proof pattern might be irritating to the practitioner, as it seems to rely on the possibility of encoding the final product — the clocked controller — already at a much earlier stage of development, namely as its own specification. But this is not completely true, as not a full behavioural description of the final product, but only its much simpler non-standard semantics is encoded by $\phi_{\mathcal{D}[\![C]\!]}$. We can even show

Lemma 8.20. *Let ϕ and ψ be DL formulae that have the same time-wise discrete models and the same free variables.*[4] *Then $(\phi, I) \in \operatorname{dom}\left(\widetilde{synt}\right)$ iff $(\psi, I) \in \operatorname{dom}\left(\widetilde{synt}\right)$, where $\widetilde{synt}$ is the synthesis procedure of theorem 8.13.*

Proof. As ϕ and ψ have the same time-wise discrete models and the same free variables, the largest timed $\mathit{free}(\phi)$-regular set contained in $\mathcal{M}[\![\phi]\!]$ is equal to the largest timed $\mathit{free}(\psi)$-regular set contained in $\mathcal{M}[\![\psi]\!]$. Hence, the construction of lemma 8.9 that constructs automata recognizing a digitization of these sets yields equivalent automata A_ϕ and A_ψ for ϕ and ψ. *synt*, itself being complete for synthesis of winning strategies for regular games, thus either succeeds on both $(I, I \setminus \mathit{free}(\phi), A_\phi)$ and $(I, I \setminus \mathit{free}(\psi), A_\psi)$ or on none of the two. As *synt* is the only partial mapping involved in $\widetilde{synt}$, this implies $(\phi, I) \in \operatorname{dom}\left(\widetilde{synt}\right)$ iff $(\psi, I) \in \operatorname{dom}\left(\widetilde{synt}\right)$. ∎

Equivalence with respect to time-wise discrete models is even coarser than logical equivalence in DL because the time-wise discrete models are a proper subset of all models. Hence, lemma 8.20 shows that the construction in the proof of theorem 8.19 guarantees existence of a rich class of specification formulae that lead to successful synthesis rather than existence of just a single, somewhat strange one.

[3]Strictly speaking, the set of models of $\phi_{\mathcal{D}[\![C]\!]}$ is slightly larger than $\mathcal{D}[\![C]\!]$, as DL cannot distinguish between almost equal trajectories (cf. lemma 5.1) such that right-continuity as present in $\mathcal{D}[\![C]\!]$ cannot be enforced. However, this closure against almost equality of trajectories is unimportant to any satisfaction property such that we may simply say that the model set $\mathcal{M}[\![\phi_{\mathcal{D}[\![C]\!]}]\!]$ and $\mathcal{D}[\![C]\!]$ are identical.

[4]Note that having the same free variables can always be achieved by adding tautologies mentioning extra variables.

8.4 Discussion

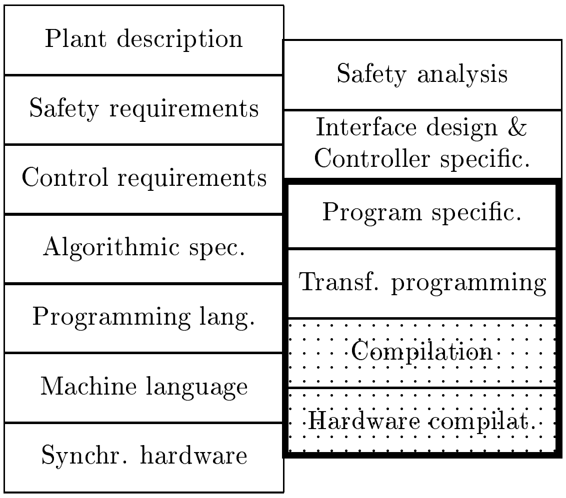
We have been able to show that below the initial development steps, which cover safety analysis, interface design, and controller specification, controller synthesis from specifications formalized in Duration Logic can be fully automated. As far as we know, this is the first positive result concerning feasibility of automatic synthesis from a dense-time Duration Calculus. Indeed, an earlier conviction in the ProCoS project has been that due to the undecidability results reported in [ZHS93], automatic translation would be impossible for almost every development step encountered in the ProCoS tower of abstractions [HHF$^+$94], except for rare special cases and for the two lowest-level refinement steps, which are compilation of an occam-like imperative programming language to machine code and to dedicated hardware (cf. figure 8.1 on the next page).

Instrumental to our success has been the insight that a common proof pattern of undecidability results for real-time and hybrid-system formalisms relies on artefacts of the formalization rather than on an encoding of inherent complexity of the design problem. In a nutshell, those proofs rely on encoding infinite information in compact Cartesian products of state and time, whereas only a finite part thereof can become effective in any embedded control application due to the finite capacity of sensor and actuator channels. This means that these undecidability results depend on distinguishing between some trajectories which would be causally equivalent within any embedded control application. An immediate consequence of that lacking discriminative power of the controlled environments is that much weaker implementation relations than trajectory inclusion, which assumes that any small-scale differences between trajectories are indeed observable, are sound. By accordingly weakening our implementation relation we have been able to simplify the implementation problem for controller specifications so far that automatic synthesis has become possible.

Although the results presented here are completely new in that they deal with synthesis for a dense-time Duration Calculus, they are not unrelated to methods that have been investigated in the ProCoS project. M.R. Hansen, J.-U. Skakkebæk, and P. Sestoft have studied decision procedures for Duration Calculus on a discrete time domain [SS94, Han94b] and, based on this, M.R. Hansen and E.-R. Olderog have been looking at the feasibility of automatic synthesis for that discrete-time Duration Calculus [HO93]. Syntactically, the subset of Duration Calculus they dealt with is identical to the one we investigated.[5] I.e., the decidability and synthesis results coincide with respect to the constructs of DC that are covered, and, furthermore, both approaches even share the same, non-elementary complexity. The major difference is that the notion of validity we have dealt with is that of dense-time Duration Calculus such that synthesis for dense-time Duration Calculus has become feasible, whereas discrete-time Duration Calculus has — due to changes in the meaning of temporal operators resulting from quantification on a discrete instead of a dense domain — a significantly different semantics.

Furthermore, our research shares the technique of exploiting constraints of the implemen-

[5] Strictly speaking, their subset is syntactically even a little more restricted, but the remaining formulae are encodable in the discrete-time setting with its richer semantic equivalence. See the discussion of discrete-time DC in section 6.5.



The ProCoS-tower of abstraction levels (left) and its associated staircase of transformations between levels (right).

The shaded transformations are known to be automatable with respect to trajectory inclusion as correctness criterion. The thickly boxed part is covered by the synthesis method described in this chapter such that automatic translation can now start from an abstract controller specification in Duration Calculus instead of an imperative program.

Figure 8.1: The ProCoS-tower of abstractions

tation technology for obtaining a decidable satisfaction problem with both Zhou Chaochen's, Zhang Jingzhong's, Yang Lu's, and Li Xiaoshan's as well as A. Bouajjani's, Y. Lakhnech's, and R. Robbana's work on deciding the satisfaction problem between specifications and implementations for appropriately chosen subsets of Duration Calculus [ZZYL94, BLR95]. However, both research groups are interested in deciding strict trajectory inclusion and therefore have to considerably restrict the set of viable specification formulae. By weakening the refinement relation, we have overcome this problem, at the only price of a little stricter constraints on the timing of control specifications. As a result, we obtain not only decidability, but even a synthesis method.

The only other synthesis techniques for real-time embedded systems that we are aware of are by H. Wong-Toi and G. Hoffmann [WTH92], and by E. Asarin, O. Maler, A. Pnueli, and J. Sifakis [MPS95, AMP95]. Both approaches use timed automata both as source and as target language of the synthesis. This simplifies matters, as both levels are closely related. Furthermore, generating timed automata, these synthesis procedures yield very abstract implementations that call for continuous clocks. It is not clear how these devices could be implemented in practice, as Maler, Pnueli, and Sifakis admit in [MPS95]:

> "We assumed that the *controller can precisely observe* the whole configuration of the plant, including *the values of all the relevant clocks.*[6] In realistic situations

[6] Maler, Pnueli, and Sifakis assume that the clocks are part of the plant that is observed, not of the controller. This simplifies the presentation, as clocks are directly reset by the plant evolution instead of upon observation of plant evolution. However, the difference between the two approaches is marginal, and in both cases the question remains how a controller can be implemented that has transitions the enabledness of which depends on the exact reading of continuous clocks.

> the plant can be observed only up to some equivalence relation on its states, and the controller has to operate under some uncertainty."
>
> [MPS95, p. 242]

An apparent solution for implementing transitions triggered by continuous clocks on synchronous hardware might be to enable the implementing transition in all those integer time instants that the original transition was enabled in. However, this procedure can introduce deadlocks, as the original transition may be enabled in a non-empty time interval that does not cover an integer time instant, or as later transitions may become blocked if the earlier transitions are forced to happen at integer time instants. As a deadlocked controller is not input-open and thus does not solve any control problem, this apparent solution is unsound, and the quest for more elaborate implementation techniques is currently open. It is not unlikely that soundness and completeness issues become intricate with such a two-level synthesis approach that first synthesizes a timed automaton from a control problem and then a discretely clocked device from the timed automaton.

In contrast to synthesis procedures yielding timed automata, our procedure yields synchronously clocked controllers, which are finite automata advancing at a constant rate of transitions. This notion of discretely clocked devices is directly supported by commercial hardware description languages like e.g. VHDL [IEEE88] or Verilog HDL [SSM$^+$93]. Both these languages feature so-called synthesizable subsets which can be automatically mapped to hardware structures like e.g. field-programmable gate arrays. As the transition tables of clocked controllers can easily be mapped to the synthesizable subsets of VHDL or Verilog HDL by just adding some syntactic sugar, our synthesis procedure immediately extends to a full hardware synthesis method. In particular, it is obvious how the devices generated by our method can be implemented in practice.

In the light of this, it is important to mention that although the different algorithms forming our synthesis procedure have not currently been integrated into a single tool, some experiments have already been made with them on realistic case studies. In particular, we have been able to generate a clocked controller directly from a DL-based *requirements specification* for the ProCoS gas-burner [RRH93]. Except for changed time constants, our requirements specification is identical to the one used in the coordinated case study of the ProCoS project. The results of this enterprise, which are reported in the next chapter, show that our synthesis procedure yields a directly hardware-implementable gas-burner controller of at least the engineering quality of the ProCoS design, thereby requiring drastically reduced design effort.

9

The Gas-Burner: An Example of Automatic Controller Synthesis from Duration Logic

The ProCoS gas-burner [RRH93, HHF$^+$94] is a case study of embedded controller development that has been developed within the ProCoS project and was selected as a collaborative effort for the second phase of the project. It is concerned with a simplistic gas-burner whose controller has to ensure

1. that it reacts upon user requests, which may ask for heat supply or for no further heat supply, and

2. that the concentration of gas in the environment is always held at a safe level.

The second requirement is deemed to be satisfied if *unignited gas may not leak from the burner for more than 4 seconds within any 30 seconds of operation.*

As the latter condition is in terms of an accumulated duration, its formalization demands considerable expressiveness. Therefore, the gas-burner case study has been adopted by the formal methods community as one of the more ambitious benchmark problems for real-time formalisms. Indeed, the problem proved to be difficult enough to serve as *the* coordinated case study of the ProCoS project, demonstrating the applicability and consistency of the ProCoS approach. Consequently, we also adopted the gas burner problem as a benchmark for the methods developed in this thesis, tackling it with the synthesis procedure proposed in chapter 8. The results are encouraging, given that even a mostly manual application of the synthesis procedure (tool support is not yet available) solved the problem within two man-days, yielding a clocked controller directly from the requirements specification, whereas the ProCoS project invested multiple man-month[1] into stepwise refinement of its coordinated case study.

[1] An exact figure of the effort invested by the ProCoS consortium cannot be given as the case study was just one research topic among others, mostly performed by the same researchers.

9.1 The requirements specification

The ProCoS gas-burner is a simple model of a computer-controlled gas-burner, depicted in figure 9.1. The embedded controller has just three binary control lines connected to the environment: *hr* signals heat requests from a thermostat, *fl* signals whether the flame is burning, and *gas* controls the gas valve. The gas valve is the only actuator in the system, and gas is expected to usually ignite spontaneously once the gas valve is opened.[2] However, gas may sometimes fail to ignite, leading to an increasing concentration of flammable gas in the environment, which is an obvious risk. The task of the controller is to prevent unsafe gas concentration in the environment through detection of ignition failures and appropriate actions, and to deliver service as required by *hr* if ignition works as expected.

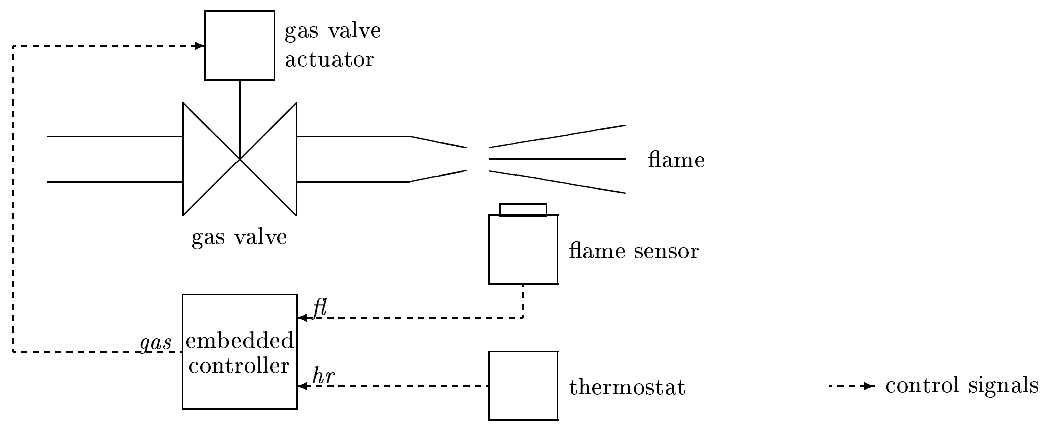


Figure 9.1: The ProCoS gas-burner

The corresponding requirements can be easily formalized using DL. Indeed, the patterns occurring have been the key motivation for development of the Duration Calculus. In the following, we stick to the original requirements given in [RRH93, BFOR93], but sometimes reduce the time constants in order not to overburden the presentation with extremely large automata occurring during synthesis.

The foremost requirement the controller has to ensure is that the gas concentration in the environment is kept below flammable level. As sensors for directly detecting the gas concentration are expensive, the gas concentration has to be safely estimated from the length and temporal distance of periods of leakage of unignited gas to the environment. We assume that safety engineers have shown that the system is safe if unignited gas may not

[2] The reader reluctant to the idea of spontaneous ignition may equally well think of an ignition device being coupled to the gas valve such that both can be simultaneously controlled by the single control signal *gas*.

leak from the burner for more than 3 seconds within any 6 seconds of operation.[3] Using DL, this can be formalized as

$$safe \stackrel{\text{def}}{=} \Box\,(l < 6 \Rightarrow \textstyle\int leak < 3) \quad .$$

Using the available sensors, leakage — or indeed a sufficient approximation of leakage — is detected through observing the flame sensor when the gas valve is open: gas is deemed to be leaking iff the flame sensor senses that the flame is not burning while the gas valve is open.

$$leak \stackrel{\text{def}}{=} gas \wedge \neg fl \quad .^{4}$$

Requirement *safe* alone is easily satisfied: as leaks can only occur when the gas valve is open, a controller permanently setting control line *gas* to false and thus keeping the gas valve closed will satisfy *safe*. However, a customer will not be satisfied with a gas-burner never delivering service. Therefore, some requirements concerning controllability of the system through *hr* are added. First, we require that $\neg hr$ will shut the gas supply within one second:

$$stop \stackrel{\text{def}}{=} \lceil \neg hr \rceil^{1} \leadsto \lceil \neg gas \rceil \quad .$$

Furthermore, we would like to require that *hr* leads to heat supply within a reasonable time span. However, this demand can only be realized if gas does not fail to ignite after opening the valve. Therefore, the startup requirement is relative to an environment assumption which formalizes the normal ignition behaviour. The normal ignition behaviour is that gas ignites soon after opening the valve such that the flame sensor reports a burning flame within 2 seconds. Whenever this is the case, heat should be supplied after at most 8 seconds of continuous heat request:

$$\begin{aligned} start &\stackrel{\text{def}}{=} flame\ ok \Rightarrow \Big(\lceil hr \rceil^{8} \leadsto \lceil fl \rceil\Big) \quad , \\ flame\ ok &\stackrel{\text{def}}{=} \lceil gas \rceil^{2} \leadsto \lceil fl \rceil \quad . \end{aligned}$$

The requirement to be guaranteed by the embedded controller then is the conjunction of above three requirements:

$$GBReq = safe \wedge stop \wedge start \quad .$$

Furthermore, the design has to respect the signature imposed by the application, namely that *hr* and *fl* are inputs to the controller and that *gas* is an output. The control problem to be solved thus is $(GBReq, \{hr, fl\})$.

[3] In the original formulation of the problem, the corresponding figures where a maximum of 4 seconds leak time within 30 seconds, as also stated on page 87.

[4] Note that this definition contains an implicit assumption that the gas valve will never fail. If the gas valve may remain open despite a closing signal $\neg gas$ then leakage cannot be detected without further sensors. However, the primary objective of the ProCoS gas-burner case-study has been to contribute to the development of real-time systems, not to deal with issues of fault-tolerance.

9.2 Deriving the case automaton

In order to automatically construct an embedded controller solving the control problem $(GBReq, \{hr, fl\})$ we will follow the synthesis procedure proposed in chapter 8, algorithm 8.14. In its first step, this algorithm requires construction of a case automaton A_{GBReq} that describes the winning positions of the controller in the associated regular game. The construction starts with calculating a finite automaton recognizing the time-wise discrete observations satisfying $\neg GBReq$. This can be done effectively using the construction of section 6.3 with $V = \{hr, fl, gas\}$. As this construction is effective, it could (and should) be mechanized. However, at the time being we have applied the rules manually, obtaining the following regular languages:

Lemma 9.1. $\bigcup_{r\in[0,1)} \mathcal{L}_{0,r}(\neg safe) = \alpha^* \cdot \left(\alpha^{\leq 6} \setminus \left(\left(\widetilde{\neg leak}\right)^* \cdot \alpha\right)^{\leq 3}\right) \cdot \alpha^*.$

Proof. By systematic application of the construction of section 6.3, exploiting equivalences of regular languages underway for simplification. ∎

Lemma 9.2. $\bigcup_{r\in[0,1)} \mathcal{L}_{0,r}(\neg stop) = \alpha^* \cdot \left(\widetilde{\neg hr} \cdot \widetilde{gas}\right) \cdot \alpha^*.$

Proof. Again through straightforward use of the construction of section 6.3. ∎

Lemma 9.3. $\bigcup_{r\in[0,1)} \mathcal{L}_{0,r}(\neg start) = \left(\alpha^* \cdot \left(\widetilde{hr}^8 \cdot \widetilde{\neg fl}\right) \cdot \alpha^*\right) \setminus \left(\alpha^* \cdot \left(\widetilde{gas}^2 \cdot \widetilde{\neg fl}\right) \cdot \alpha^*\right).$ ∎

Corollary 9.4.

$$\bigcup_{r\in[0,1)} \mathcal{L}_{0,r}(\neg GBReq) = \left(\begin{array}{cc} & \alpha^* \cdot \left(\alpha^{\leq 6} \setminus \left(\left(\widetilde{\neg leak}\right)^* \cdot \alpha\right)^{\leq 3}\right) \cdot \alpha^* \\ \cup & \alpha^* \cdot \left(\widetilde{\neg hr} \cdot \widetilde{gas}\right) \cdot \alpha^* \\ \cup & \left(\alpha^* \cdot \left(\widetilde{hr}^8 \cdot \widetilde{\neg fl}\right) \cdot \alpha^*\right) \setminus \left(\alpha^* \cdot \left(\widetilde{gas}^2 \cdot \widetilde{\neg fl}\right) \cdot \alpha^*\right) \end{array}\right).$$

Proof. By lemma 6.16, $\bigcup_{r\in[0,1)} \mathcal{L}_{0,r}(\neg GBReq) = \left(\begin{array}{cc} & \bigcup_{r\in[0,1)} \mathcal{L}_{0,r}(\neg safe) \\ \cup & \bigcup_{r\in[0,1)} \mathcal{L}_{0,r}(\neg stop) \\ \cup & \bigcup_{r\in[0,1)} \mathcal{L}_{0,r}(\neg start) \end{array}\right)$. The result follows with the previous three lemmata. ∎

Having obtained a regular expression describing $\bigcup_{r\in[0,1)} \mathcal{L}_{0,r}(\neg GBReq)$, we used the AMoRE tool of O. Matz, A. Miller, A. Potthoff, W. Thomas, and E. Valkema [MMP+95] to generate a minimal deterministic automaton A_{GBReq} recognizing the regular language $(\{hr, fl, gas\} \to \mathbb{B})^* \setminus \bigcup_{r\in[0,1)} \mathcal{L}_{0,r}(\neg GBReq)$. The result was an automaton with 196 states the first thirty of which are shown in table 9.1 on the next page.

State	Status	Transitions							
		$\neg gas$				gas			
		$\neg hr$		hr		$\neg hr$		hr	
		$\neg fl$	fl	$\neg fl$	fl	$\neg fl$	fl	$\neg fl$	fl
q0		q159	q68	q162	q47	q72	q72	q72	q72
q1	final	q191	q191	q48	q48	q195	q192	q73	q59
q2		q132	q132	q40	q40	q140	q132	q46	q43
q3	final	q72	q72	q72	q72	q72	q72	q72	q72
q4	final	q132	q189	q40	q41	q140	q189	q46	q44
q5	final	q189	q189	q11	q11	q186	q189	q25	q77
q6	final	q159	q191	q162	q65	q56	q192	q53	q59
q7	final	q68	q191	q47	q48	q54	q192	q55	q59
q8		q68	q68	q47	q47	q54	q71	q55	q60
q9		q159	q68	q162	q47	q56	q71	q53	q60
q10		q68	q68	q67	q67	q72	q71	q72	q70
q11	final	q189	q189	q36	q36	q186	q189	q1	q78
q12	final	q189	q189	q39	q39	q3	q189	q3	q87
q13	final	q29	q29	q169	q169	q72	q72	q72	q72
q14		q132	q132	q42	q42	q72	q132	q72	q134
q15	final	q68	q191	q67	q69	q72	q192	q72	q138
q16	final	q191	q191	q102	q102	q3	q192	q3	q104
q17	final	q191	q191	q69	q69	q3	q192	q3	q138
q18	final	q159	q191	q162	q65	q3	q192	q3	q59
q19		q29	q132	q169	q42	q3	q135	q3	q136
q20	final	q191	q191	q48	q48	q3	q192	q3	q59
q21	final	q159	q191	q162	q48	q56	q192	q53	q59
q22	final	q68	q191	q47	q48	q72	q192	q72	q59
q23	final	q159	q191	q162	q48	q3	q192	q3	q59
q24	final	q191	q191	q65	q65	q195	q192	q73	q59
q25	final	q191	q191	q65	q65	q3	q192	q3	q59
q26	final	q189	q189	q37	q37	q3	q189	q3	q83
q27	final	q189	q189	q117	q117	q3	q190	q3	q115
q28	final	q191	q191	q103	q103	q195	q192	q155	q61
q29	final	q29	q29	q178	q178	q72	q72	q72	q72
				⋮					
q167	initial, final	q189	q189	q170	q170	q186	q189	q150	q188
				⋮					

Table 9.1: The case automaton A_{GBReq} constructed in step 1 of algorithm 8.14

9.3 Synthesis of the synchronous controller

Determining whether the control problem $(GBReq, \{hr, fl\})$ is solvable and, if so, synthesizing an appropriate controller is extremely simple. As the non-accepting states of A_{GBReq} represent situations where something undesirable has happened, a successful control strategy has to avoid being driven into these states. Therefore, all states from which the environment may drive the automaton to an accepting state regardless of the sequence of *gas* and $\neg gas$ events, i.e. regardless of how the output is controlled, have to be marked as uncontrollable. I.e., the set U of uncontrollable states to be marked is the smallest set of states that

1. contains all non-final states, and
2. contains any state σ that satisfies

$$\exists i \in \{hr, fl\} \to \mathbb{B} \,.\, \forall o \in \{gas\} \to \mathbb{B} \,.\, \Big((\sigma, i \oplus o, \sigma') \in T \Rightarrow \sigma' \in U\Big) \quad ,$$

   where T is the transition relation, i.e. contains any state from which the environment can drive the system to a state in U.

The set U of uncontrollable states can be computed by the procedure of step 2 of algorithm 8.14, which incrementally marks those states that can be driven to a non-final state in at most j steps by the environment. Table 9.2 on the facing page shows the transition table of A_{GBReq} after marking the uncontrollable states.

The system is controllable iff the initial state does not become marked 'uncontrollable' as this means that from the initial state, i.e. the start of time, there always is a possibility to prevent being driven to a non-accepting state, i.e. to an undesirable situation, by the environment. As `q167` has not been marked, the control problem $(GBReq, \{hr, fl\})$ has a solution. Thus step 3 of algorithm 8.14 is passed.

According to step 4 of the algorithm, a winning strategy for the controller is then obtained by replacing the set of final states by the set of controllable states. I.e., the generated control strategy is to always take a transition leading to an unmarked and thus controllable state. The resulting automaton can be mapped to a clocked controller by applying the mapping $Aut^{-1}(\cdot)$ after identifying all non-final, i.e. uncontrollable, states. By definition of $Aut(\cdot)$ this simply amounts to removing all the uncontrollable states and all transitions leading to them. The resulting transition table, which is shown in table 9.3 on page 94, is a sound clocked controller solving the control problem $(GBReq, \{hr, fl\})$. However, a glance at table 9.3 on page 94 reveals that there is still a lot of nondeterminism in that control strategy that could be removed, yielding a simpler controller. E.g. initially, i.e. from state q167, the controller may upon input $hr \wedge \neg fl$ either switch *gas* off and go to state q170, or it may switch *gas* on and go to state q150. It is worth noting that such nondeterminism may be resolved arbitrarily without risking unsoundness: every choice between the available output/successor-state combinations provides a correct control strategy, independent of what criterion is used for selection. Therefore, we may select arbitrary heuristics for resolving the nondeterminism: we may cautiously prefer to switch off *gas* whenever this is possible, or we may aggressively switch it on whenever possible, or we may base our selection on the circuit cost required for determining output and successor states.

State	Status	Transitions							
		$\neg gas$				gas			
		$\neg hr$		hr		$\neg hr$		hr	
		$\neg fl$	fl	$\neg fl$	fl	$\neg fl$	fl	$\neg fl$	fl
q0	unctrl.	q159	q68	q162	q47	q72	q72	q72	q72
q1	final	q191	q191	q48	q48	q195	q192	q73	q59
q2	unctrl.	q132	q132	q40	q40	q140	q132	q46	q43
q3	unctrl., fin.	q72	q72	q72	q72	q72	q72	q72	q72
q4	unctrl., fin.	q132	q189	q40	q41	q140	q189	q46	q44
q5	final	q189	q189	q11	q11	q186	q189	q25	q77
q6	final	q159	q191	q162	q65	q56	q192	q53	q59
q7	unctrl., fin.	q68	q191	q47	q48	q54	q192	q55	q59
q8	unctrl.	q68	q68	q47	q47	q54	q71	q55	q60
q9	unctrl.	q159	q68	q162	q47	q56	q71	q53	q60
q10	unctrl.	q68	q68	q67	q67	q72	q71	q72	q70
q11	final	q189	q189	q36	q36	q186	q189	q1	q78
q12	unctrl., fin.	q189	q189	q39	q39	q3	q189	q3	q87
q13	final	q29	q29	q169	q169	q72	q72	q72	q72
q14	unctrl.	q132	q132	q42	q42	q72	q132	q72	q134
q15	unctrl., fin.	q68	q191	q67	q69	q72	q192	q72	q138
q16	final	q191	q191	q102	q102	q3	q192	q3	q104
q17	unctrl., fin.	q191	q191	q69	q69	q3	q192	q3	q138
q18	final	q159	q191	q162	q65	q3	q192	q3	q59
q19	unctrl.	q29	q132	q169	q42	q3	q135	q3	q136
q20	unctrl., fin.	q191	q191	q48	q48	q3	q192	q3	q59
q21	final	q159	q191	q162	q48	q56	q192	q53	q59
q22	unctrl., fin.	q68	q191	q47	q48	q72	q192	q72	q59
q23	final	q159	q191	q162	q48	q3	q192	q3	q59
q24	final	q191	q191	q65	q65	q195	q192	q73	q59
q25	final	q191	q191	q65	q65	q3	q192	q3	q59
q26	unctrl., fin.	q189	q189	q37	q37	q3	q189	q3	q83
q27	final	q189	q189	q117	q117	q3	q190	q3	q115
q28	final	q191	q191	q103	q103	q195	q192	q155	q61
q29	final	q29	q29	q178	q178	q72	q72	q72	q72
				⋮					
q167	initial, final	q189	q189	q170	q170	q186	q189	q150	q188
				⋮					

Table 9.2: The case automaton A_{GBReq} with uncontrollable states marked.

State	Status	Transitions							
		$\neg gas$				gas			
		$\neg hr$		hr		$\neg hr$		hr	
		$\neg fl$	fl	$\neg fl$	fl	$\neg fl$	fl	$\neg fl$	fl
q1		q191	q191	q48	q48	q195	q192	q73	q59
q5		q189	q189	q11	q11	q186	q189	q25	q77
q6		q159	q191	q162	q65	q56	q192	q53	q59
q11		q189	q189	q36	q36	q186	q189	q1	q78
q13		q29	q29	q169	q169	q72	q72	q72	q72
q16		q191	q191	q102	q102	—	q192	—	q104
q18		q159	q191	q162	q65	—	q192	—	q59
q21		q159	q191	q162	q48	q56	q192	q53	q59
q23		q159	q191	q162	q48	—	q192	—	q59
q24		q191	q191	q65	q65	q195	q192	q73	q59
q25		q191	q191	q65	q65	—	q192	—	q59
q27		q189	q189	q117	q117	—	q190	—	q115
q28		q191	q191	q103	q103	q195	q192	q155	q61
q29		q29	q29	q178	q178	q72	q72	q72	q72
⋮									
q167	initial	q189	q189	q170	q170	q186	q189	q150	q188
⋮									

Table 9.3: Nondeterministic transition table satisfying *GBReq*

For simplicity, we used the first option and removed any transition switching *gas* on that has a corresponding transition switching *gas* off. I.e. if under some input i both $i \oplus (gas \mapsto \texttt{true})$ and $i \oplus (gas \mapsto \texttt{false})$ lead to controllable successor states then we simply removed the transition for $i \oplus (gas \mapsto \texttt{true})$. As this procedure removes a major part of the transitions, it resulted in a transition table with only 29 reachable states. Furthermore, the resulting transition table is deterministic such that automaton minimization, yielding the corresponding minimal *deterministic* transition table, may not increase table size. Thus, minimization was applied after resolving the nondeterminism, yielding a control automaton with only 10 states, which is depicted in figure 9.2 on the next page.

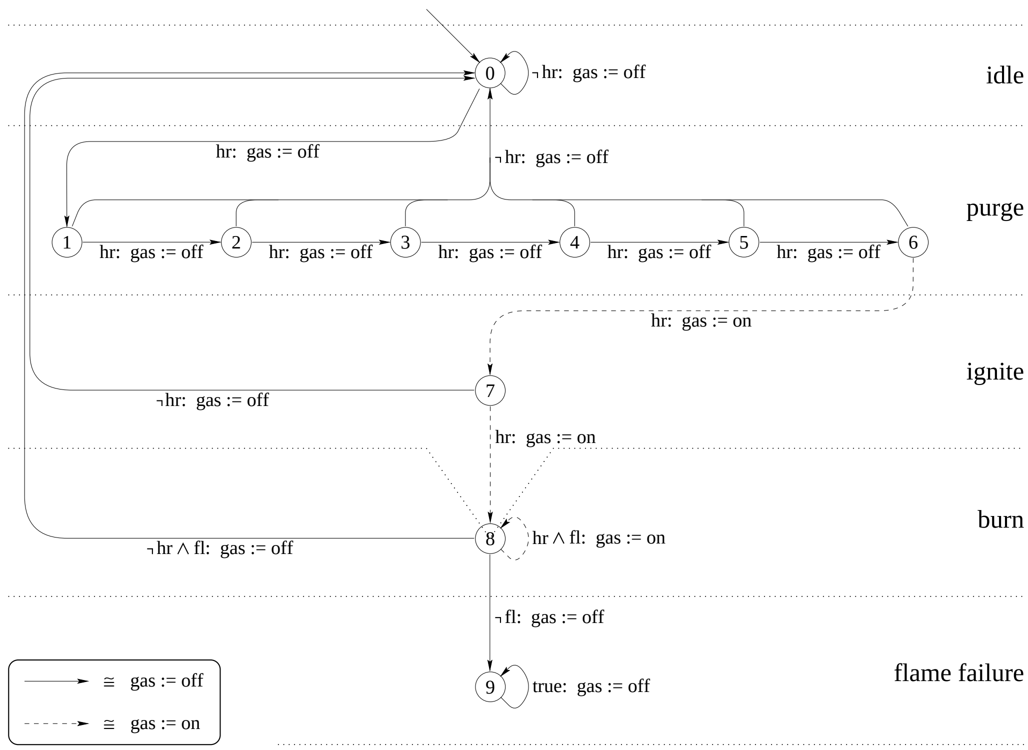
We were surprised to see that the resulting controller, although generated by a fully effective procedure, even resembles the phase structure of the manually developed controllers of A. P. Ravn, H. Rischel, and K. M. Hansen [RRH93]. Ravn, Rischel, and Hansen developed their control skeleton around the idea of an *idle phase*, where the controller waits for the next heat request, a *purge phase*, where the gas concentration in the environment is reduced through keeping gas shut off for a while before doing an ignition attempt, an *ignite phase* opening the valve long enough to get the flame burning if *flame ok* holds, and a *burn phase* being entered once the flame is stably burning and being left if it either extinguishes or heat request *hr* is withdrawn. See figure 9.2 on the following page for more details of the phase design of [RRH93] and how these phases can be identified in the automatically generated gas-burner controller.

9.4 Discussion

The results of this very first attempt in automatic synthesis of embedded controllers from a dense-time Duration Calculus are very promising, both concerning the effort invested and the engineering quality of the results obtained. The engineering quality of the delivered product is directly comparable to that delivered by the combined invent-and-verify and step-wise refinement approach of ProCoS, as the synthesized controller even reveals the phased design of its ProCoS counterpart. Concerning the end-product, our design is even more efficient since the clocked controller may be directly mapped to a synchronously clocked hardware device, whereas the ProCoS project took the digression to map the control automaton to an occam-like language, thereby adding quite a bit of handshaking circuitry for synchronous channels and for occam's composition operators to the final hardware device.

The effort we invested into this case study was two man-days, and could be considerably lower if tool support were available for generating the case automaton $A_{\neg GBReq}$ and analyzing its transition table. Currently, only the transformation from the regular language describing the time-wise discrete models of $\neg GBReq$ to a corresponding minimal deterministic finite automaton has been done with tool support, whereas calculation of the regular languages was a completely manual effort. But even the current figure of two man-days compares favorably to multiple man-month invested by the ProCoS project into the gas-burner case study.

However, although promising, the findings made on just the gas-burner case study are not sufficient to safely judge the versatility of the synthesis method. Many more case studies



The automatically synthesized gas burner resembles the phase design of the manually developed gas burners of [RRH93, BFOR93]. The corresponding phase names used in those designs are indicated on the right. A new phase label 'flame failure', which was not clearly differentiated in the manual development, has been added.

The controller starts in an *idle* phase, where it waits for a heat request while gas is kept turned off. Once a heat request is encountered, the controller enters a *purge* phase which turns gas off for six more time units, allowing unignited gas to vapor from the environment surrounding the boiler. Thereafter, gas can be turned on for at least two time units without risking a violation of the safety requirement. If the heat request still prevails, the *ignite* phase is entered, which turns on the gas for two time units unless heat request is withdrawn during that period. Thereafter, the flame sensor is checked. If the flame is not burning after the ignition phase then the physical assumption *flame ok* is not met by the environment. Detecting this, the controller goes to phase *flame failure* and quits service, turning off gas forever, as *flame ok* is a precondition of requirement *start*. If the flame is burning after the *ignite* phase then the controller enters the *burn* phase, keeping gas flow on until heat request is withdrawn, which sends the controller back to the *idle* phase to wait for the next heat request, or until the $\neg fl$ signals a violation of *flame ok* which again results in quitting service.

Note that state 8 is shared between the *ignite* and the *burn* phase, belonging to the ignite phase, where gas is kept on independently of the reading of the flame sensor, upon first entry and belonging to the *burn* phase, where gas is kept on only if the flame sensor signals a burning flame, upon repeated entry.

Figure 9.2: The synthesized gas-burner controller

have to be done before we may claim practical applicability of the method. Nevertheless, the current success has far-reaching implications for future research. First, it suggests that it should be worthwhile to invest effort in tool support for Duration Logic over time-wise discrete models. Second, it demonstrates that very abstract and even undecidable temporal logics like Duration Calculus could serve as source languages in hardware synthesis. The implications of this result go far beyond Duration Calculus. Techniques based on analyzing observability constraints imposed by interface circuitry like in chapter 8 should be able to pave the way for hardware synthesis methods based on other temporal logics which currently seem out of scope of automatic synthesis.

Part III

Key-Press Verification of Asynchronous Circuits

Using a rigorous notion of implementation correctness, soundness of a discrete-state model of asynchronous hardware is formally verified against a simple continuous-state, circuit-level model. It is demonstrated how reactivity constraints of the components justify augmenting additional postulates to the discrete-state model. While automatically model-checking the original discrete-state semantics against $\{\lceil P \rceil, l = k\}$-formulae is in general impossible, the additional postulates suffice for gaining decidability. As the additional postulates may be added at no cost, this evidences that undecidability is sometimes caused by a particular semantic abstraction rather than by inherent complexity of the application domain.

10

Test Preorder and Refinement

Given that in chapter 8 mere exploitation of sensor latencies has provided — for the first time ever — a controller synthesis procedure for a dense-time Duration Calculus, a fundamental study of the underlying argument seems in place. By now, just the lacking discriminative power of interface circuitry has been taken into account. As any component in any technical device is subject to similar constraints in observing the environment as sensors and actuators are, the gains from a more general application of the observations made in chapter 2 might be serious.

In this chapter, we pave the way for this by showing how such observability constraints justify much weaker refinement relations than trajectory inclusion. The key concept, explained on an example of real-world refinement reasoning, is a generalization of testing preorders [Hen88]. This turns out to be a natural notion of refinement as differences between devices are only relevant to design if there is an application context where they manifest. From this characterization of refinement, sound and complete proof rules for showing that a system is a refinement of another and for compiler correctness are given that do not require an explicit construction of the testing preorder.

As the proposed refinement relation, when applied to trajectory-based semantics, is in general much weaker than trajectory inclusion, these proof rules allow to rigorously verify refinements that are well beyond the trajectory-based model of refinement. In chapter 12, we will exploit this for verifying an implementation of digital systems by continuously operating networks of MOS-transistors, thereby bridging an often perceived 'semantic gap' between the discrete-state formal models used in hardware verification and the actual implementation.

10.1 Observation-oriented semantics and refinement in engineering

In research on concurrent systems it has become popular to use concurrency models based upon physical analogies like state and trajectory in order to have a smooth interface to more general models encountered in more integrated engineering enterprises, particularly those concerning physical reality. Trajectory-based models are intended to reflect the signal

The material covered by this chapter has previously been available to the public as a ProCoS report [Frä94].

waveforms that could be expected from probing the system with an oscilloscope or logic analyzer, and have therefore also been termed 'observation-oriented semantics'. Their close resemblance of physical measurements is thought to ease their use in engineering contexts, providing engineers with a direct understanding of formal concepts like refinement. However, this promise is hard to meet in practice, as we will see on an example.

Up-to-now, the only generally agreed notion of refinement in observation-oriented semantics is inclusion of the set of possible trajectories of the refining system in that of the refined system. In the various temporal logics widely applied for describing the possible trajectories of such systems (e.g. [MP92, Lam91, Mos85, ZHR91]), this usually corresponds to validity of an implication $\pi \Rightarrow \psi$, where π is the formula describing the refining system and ψ a formula describing the refined system.[1] Unfortunately, the same generality of this refinement relation that makes it universally acceptable requires it to be more discriminating than actually necessary in specific applications. The standard technique to partly compensate for this effect is to impose closure conditions on the trajectory sets which remove some of the distinguishing power. But as this is quite artificial and as the effects on the formalization of the individual operators are sometimes complex, this may be in certain applications as inconvenient from an engineering point of view as an overly restrictive refinement relation.

E.g., consider an electrical engineer developing electro-acoustic circuitry who has the choice between two amplifier modules with the same gain and frequency response in the audible range, but different harmonic distortion levels. Although the engineer will clearly consider the less distorting amplifier a valid replacement for (i.e. a refinement of) the other one, it does not directly (i.e. in a standard model without additional closures imposed) qualify as such in the implicational sense of refinement: upon the same input signal the amplifier modules produce simply different output trajectories, hence they have incomparable (in the sense of set inclusion) trajectory sets. To see this just notice that (if we idealize and ignore noise) both amplifiers are deterministic devices and thus produce exactly one output trajectory per input signal. Hence, they are either equal (which is obviously false) or incomparable.

As mentioned, the technique proposed in so-called observation-oriented semantics to remedy this lack of refinement capabilities is to downwardly close denotations with respect to expectations. That is, here, to add *all* trajectories which show at most the distortion level of the amplifier under consideration to its denotation. This results in the desired refinement property, but is not as straightforward as it may seem at first glance. Whether this closure is reasonable depends strongly on the set of devices the engineer is allowed to connect to the amplifiers and on the shape of expectations on the complete device to be built.

To see this just notice that there are contexts where the proposed closure is too coarse to provide sufficient reasoning power: it discards all information on the spectral distribution of distortion making estimation of distortion level impossible if some amplifiers with frequency-dependent gain are incorporated in the complete device. On the other hand this does not

[1] Note that we only talk about complete systems here. More liberal refinement concepts have been proposed for components of systems (e.g. rely-guarantee paradigms, hiding mechanisms for internal detail, etc.), but the rationale of their soundness has commonly been trajectory inclusion for the complete systems.

mean that the downward closure on distortion should be always performed in a way which preserves its spectral distribution. In most cases this will be far too restrictive, particularly when gain is frequency-independent (within the audible range) for all devices and overall requirements are only concerned with *total* harmonic distortion (an integrated measure through the frequency range).

The lesson to be learned from this is that it is by no means simple to find adequate closures of denotations for refinement, and that the adequacy of a certain closure may crucially depend on the available operators of the language and the shape of the overall requirements which may be placed on complete systems. Hence — as the closure is part of the semantic model — slight changes in the language may require drastic changes in its semantics.

In this chapter we propose to refrain from encoding such contextual information in the semantic model. Instead, the most direct semantic models at hand may be used (e.g. circuit models of electrical engineering) and implementation freedom may be gained by using more liberal refinement concepts than trajectory inclusion. The emphasis will be on giving a precise condition on whether a relation is an acceptable refinement relation and on a proof rule for this condition.

For establishing a general condition for a relation between components being a sound refinement relation we should analyze the engineer's reasoning that makes him so sure that he may replace an amplifier by a less distorting counterpart:

It starts from the fact that an engineering product is designed to *satisfy conditions* on its overall behaviour, some of which are naturally downward closed. One such class of downward closed conditions are conditions on total harmonic distortion: if 1% of total harmonic distortion is acceptable then the engineer may always deliver a device with 0.7% of total harmonic distortion. He will never be faced with a minimum requirement on distortion. Furthermore, over a wide range of devices, there is monotonicity with respect to distortion: Decreasing distortion of a subcomponent will not increase distortion of the complete device. Hence, in such assemblies, whatever the requirements on the total system may be, above replacement of amplifiers is sound as it will not invalidate satisfaction of any condition that might have been placed on the complete design.

More formally, this reasoning can be captured as:

> Given a set S of possible conditions on systems[2] and a satisfaction relation `sat` between systems and conditions in S, a *device P refines a device Q*, iff in all contexts $c(.)$ such that $c(Q)$ is a complete system, $c(P)$ becomes also a complete system and
>
> $$c(Q) \texttt{ sat } s \quad \Rightarrow \quad c(P) \texttt{ sat } s$$
>
> holds for each $s \in S$.

Note that this closely resembles the idea of *testing preorders* obtained from testing processes through all contexts that can be composed in parallel, as suggested and analyzed

[2]This is to be understood as being the full set of conditions that *may* ever be placed on systems, *not* as the set of conditions that is actually placed on the single, concrete system under development. Normally, the latter will be a fairly small subset of S.

by M. Hennesy for CCS-like languages [Hen88]. However, our notion of testing is more general due to three notable differences in the basic setup:

1. We allow for a multitude of success criteria instead of just one, reflecting the fact that in system engineering there is not the one success criterion encompassing all the others. As will become apparent in the next section, even if the success criteria form a lattice with respect to satisfaction, there is no hope that they can be generally collapsed to a single criterion without altering the refinement relation.

2. We do not distinguish between "must" and "may" tests. It is up to the satisfaction relation $\mathtt{sat}$ whether it establishes satisfaction in the "must" or the "may" sense or even both, if the set S of success criteria is rich enough to comprise both kinds of criteria.

3. As the connection between systems and success conditions is simply mediated by a satisfaction relation, we are free to start from any kind of semantics of systems, not being limited to semantics assigning execution paths or even being operational.

In the following, we will formalize and investigate above notion of refinement in the setting of denotational semantics. Denotational semantics seems to be a good starting point, as many languages encountered in formal models of embedded control either directly have a denotational, trajectory-based, semantics, or indirectly have by using temporal logics as a meta-language for describing the semantics, with the logics having a truth-functional semantics.

10.2 Testing and its preorders in a denotational setting

As noted in the introduction, we are particularly interested in observation-oriented semantics. There, the semantic domain is $(\mathit{Proc}, \sqsubseteq)$, where $\mathit{Proc} \subseteq \mathcal{P}(\mathit{Traj})$ is a (not necessarily proper) subset of the powerset of the possible trajectories Traj and the domain ordering $(\sqsubseteq) = (\supseteq)$ is the reverse inclusion ordering. In a linear-time setting, validity of conditions is defined on individual trajectories and hence conditions can be encoded by sets of trajectories themselves, with satisfaction being set inclusion,[3] i.e.

$$P \;\mathtt{sat}\; \phi \quad \text{iff} \quad P \subseteq \phi \;,$$

or equivalently

$$P \;\mathtt{sat}\; \phi \quad \text{iff} \quad \phi \sqsubseteq P \;.$$

This notion of satisfaction can be directly generalized to arbitrary denotational semantics, taking members of the semantic domain as conditions and reverse domain ordering as satisfaction relation. Note that this does also nicely coincide with the interpretation of the domain ordering as an information-content ordering: $\phi \sqsubseteq P$ means that P satisfies any property that ϕ describes, plus possibly some additional ones [RR94].

[3]The generalization to branching time frameworks would use singleton-sets of trees as denotations and general sets of trees as conditions, thus providing a similar framework.

Let the *language* $\mathcal{L}$ be the many-sorted term algebra (T, Ax, O) with T being the set of sorts, $Ax \in T$ be the sort of complete systems (e.g. programs), and O a set of operators with associated types in $T^* \times T$. (We will use the more intuitive notation $o : t_1, \ldots, t_n \to t$ in the remainder of this chapter to denote that o is an operator with type $(\langle t_1, \ldots, t_n \rangle, t)$.) $\mathcal{L}$-terms of sort Ax are called $\mathcal{L}$-programs. $\mathcal{I}$ is an $\mathcal{L}$-*semantics* if it associates to each sort $t \in T$ a domain $\mathcal{I}_t = (D_{\mathcal{I}_t}, \sqsubseteq_{\mathcal{I}_t})$ and to each operator $o \in O$ with type $(\langle t_1, \ldots, t_n \rangle, t)$ a monotonic function $\mathcal{I}_o : \mathcal{I}_{t_1} \times \ldots \times \mathcal{I}_{t_n} \to \mathcal{I}_t$. $\mathcal{I}$ extends to a semantics $\mathcal{I}[\![\cdot]\!]$ of arbitrary $\mathcal{L}$-terms in the obvious way.

Let $\mathcal{I}$ be an $\mathcal{L}$-semantics. A *criterion* s is an arbitrary element $s \in D_{\mathcal{I}_{Ax}}$. We say an $\mathcal{L}$-program π *satisfies* s, denoted $\pi\ \mathtt{sat}\ s$, iff $s \sqsubseteq_{\mathcal{I}_{Ax}} \mathcal{I}[\![\pi]\!]$. From now on let $S \subseteq D_{\mathcal{I}_{Ax}}$ be the set of *possible success criteria*, i.e. those criteria that may be posed on complete systems.

For a sort $t \in T$, a t-*context* is an $\mathcal{L}^t$-program, with $\mathcal{L}^t \stackrel{\text{def}}{=} (T, Ax, O \cup \{\sqcup\})$, where $\sqcup \notin O$ is a new operator with type $(\langle\rangle, t)$. The additional operator $\sqcup$ should be interpreted as a placeholder for objects of type t. For a t-context c and an $\mathcal{L}$-term P of sort t, $c(P)$ denotes the $\mathcal{L}$-program obtained by substituting all occurrences of $\sqcup$ in c by P.

With these preliminaries, we can define the testing preorder on $\mathcal{L}$-terms induced by semantics $\mathcal{I}$ and the set of success criteria S:

Definition 10.1 (Refinement). *Let $t \in T$, and P and Q $\mathcal{L}$-terms of sort t. P is* refined *by Q (with respect to the testing preorder induced by $\mathcal{I}$ and S), denoted $P \leq Q$, iff*

$$c(P)\ \mathtt{sat}\ s \quad \Rightarrow \quad c(Q)\ \mathtt{sat}\ s \tag{10.1}$$

for each t-context c and each $s \in S$.
By definition of satisfaction, (10.1) is equivalent to

$$s \sqsubseteq_{\mathcal{I}_{Ax}} \mathcal{I}[\![c(P)]\!] \quad \Rightarrow \quad s \sqsubseteq_{\mathcal{I}_{Ax}} \mathcal{I}[\![c(Q)]\!]\ .$$

Note that generally a non-singleton set S of success criteria cannot be replaced by a single criterion without altering refinement, as $\mathcal{I}[\![c(P)]\!]$ may be larger than one and smaller as another element of S, and at the same time even incomparable with a third, such that using more than one success criterion will normally add information to refinement, making the refinement relation more discriminating.

Note, furthermore, that we allow placeholders to occur more than once in a context. This is mainly for the sake of simplicity of the definitions, as it makes t-contexts form a context-free language, but may conflict with the reader's intuitive grasp of the engineering pragmatics that contexts are intended to model. Nevertheless, this simplicity comes at no extra cost, as the refinement relation is independent of whether its definition refers to general contexts or only to such with exactly one placeholder:

Theorem 10.2 (Multiple placeholders do not add distinguishing power). *Let P and Q be $\mathcal{L}$-terms of sort t, c an arbitrary t-context and $s \in S$. If $c(P)\ \mathtt{sat}\ s$ and $\neg\,(c(Q)\ \mathtt{sat}\ s)$ (i.e., if c and s distinguish P and Q) then there is a t-context $\tilde{c}$ with just one placeholder such that $\tilde{c}(P)\ \mathtt{sat}\ s$ and $\neg\,(\tilde{c}(Q)\ \mathtt{sat}\ s)$.*

Proof. Let P and Q be $\mathcal{L}$-terms of sort t, c a t-context and $s \in S$ s.t. $c(P)$ $\texttt{sat}$ s and $\neg\,(c(Q)$ $\texttt{sat}$ $s)$. Let $n \in \mathbb{N}$ be the number of placeholders in c.

Now, let $\mathcal{L}^{t^n} \stackrel{\text{def}}{=} (T, Ax, O \cup \{\sqcup_1, \ldots, \sqcup_n\})$, where $\sqcup_1 \notin O$ to $\sqcup_n \notin O$ are n distinct new operators with type $(\langle\rangle, t)$, and let $\bar{c}$ be an $\mathcal{L}^{t^n}$-program, where each placeholder $\sqcup_1$ to $\sqcup_n$ occurs exactly once and where substituting all those placeholders by $\sqcup$ gives c. Thus, $\bar{c}$ is a generalization of c which can be used to selectively substitute some of the placeholders by P and some by Q.

Therefore, let $\bar{c}_i$ for $i \in \{0, \ldots, n\}$ denote the $\mathcal{L}$-program which is obtained by substituting P for $\sqcup_1$ to $\sqcup_i$ and Q for $\sqcup_{i+1}$ to $\sqcup_n$ in $\bar{c}$. Obviously, $\bar{c}_n = c(P)$ and $\bar{c}_0 = c(Q)$. Hence, $\bar{c}_n$ $\texttt{sat}$ s and $\neg\,(\bar{c}_0$ $\texttt{sat}$ $s)$. Thus, $\{i \in \mathbb{N}_{\leq n} \mid \bar{c}_i \texttt{ sat } s\}$ is non-empty and its least element j is larger than 0. Now define $\tilde{c}$ as being the t-context which is obtained by substituting P for $\sqcup_1$ to $\sqcup_{j-1}$, $\sqcup$ for $\sqcup_j$, and Q for $\sqcup_{j+1}$ to $\sqcup_n$ in $\bar{c}$. $\tilde{c}$ has exactly one placeholder. Furthermore, as $\tilde{c}(P) = \bar{c}_j$ and $\tilde{c}(Q) = \bar{c}_{j-1}$, we have $\tilde{c}(P)$ $\texttt{sat}$ s and $\neg\,(\tilde{c}(Q)$ $\texttt{sat}$ $s)$. ▟

Aiming at a stepwise system development process, transitivity of the refinement relation is desirable. Furthermore, a modular approach to system development calls for the possibility of local refinements, requiring monotonicity of $\mathcal{L}$'s operators with respect to refinement as well as reflexivity of the refinement relation. $\leq$'s transitivity and reflexivity follows immediately from transitivity and reflexivity of implication. Hence

Theorem 10.3 (Refinement is a preorder). *For each sort $t \in T$, $\leq$ is a preorder on the set of $\mathcal{L}$-terms of sort t.* ▟

Furthermore, each operator $o \in O$ is monotonic with respect to $\leq$. Let $o : t_1, \ldots, t_n \to t$ be an operator in O and $P_1, \ldots, P_n$ and $Q_1, \ldots, Q_n$ be $\mathcal{L}$-terms of sorts $t_1, \ldots, t_n$.

Theorem 10.4 (Monotonicity of operators with respect to refinement). *If $P_i \leq Q_i$ for all $i \in \{1, \ldots, n\}$ then $o(P_1, \ldots, P_n) \leq o(Q_1, \ldots, Q_n)$.*

Proof. By transitivity of $\leq$ this follows immediately from monotonicity of o in each individual parameter, as expressed by the following lemma. ▟

Lemma 10.5. *Let $i \in \{1, \ldots, n\}$ and let P and Q be $\mathcal{L}$-terms of sort t_i. If $P \leq Q$ then $o(P_1, \ldots, P_{i-1}, P, P_{i+1}, \ldots, P_n) \leq o(P_1, \ldots, P_{i-1}, Q, P_{i+1}, \ldots, P_n)$.*

Proof. Assume $P \leq Q$. Let c be a t-context. Then $c(o(P_1, \ldots, P_{i-1}, \sqcup, P_{i+1}, \ldots, P_n)$ is a t_i-context and thus $P \leq Q$ implies that if $c(o(P_1, \ldots, P_{i-1}, P, P_{i+1}, \ldots, P_n))$ satisfies a success criterion s then so does $c(o(P_1, \ldots, P_{i-1}, Q, P_{i+1}, \ldots, P_n))$. As c is an arbitrary t-context this shows that $o(P_1, \ldots, P_{i-1}, P, P_{i+1}, \ldots, P_n) \leq o(P_1, \ldots, P_{i-1}, Q, P_{i+1}, \ldots, P_n)$. I.e., o is monotonic in its i-th argument with respect to the testing preorder. ▟

Furthermore, testing preorder can be shown to cover denotational order:

Theorem 10.6 (Denotational order implies refinement). *Let P and Q be $\mathcal{L}$-terms of sort $t \in T$. If $\mathcal{I}[\![P]\!] \sqsubseteq_{\mathcal{I}_t} \mathcal{I}[\![Q]\!]$ then $P \leq Q$.*

Proof. By monotonicity of all operators $o \in O$ with respect to the denotational order, $\mathcal{I}[\![P]\!] \sqsubseteq_{\mathcal{I}_t} \mathcal{I}[\![Q]\!]$ implies $\mathcal{I}[\![c(P)]\!] \sqsubseteq_{\mathcal{I}_{Ax}} \mathcal{I}[\![c(Q)]\!]$ for any t-context c. Hence, by transitivity of $\sqsubseteq_{\mathcal{I}_{Ax}}$, $c(P)$ `sat` s implies $c(Q)$ `sat` s. ∎

On one hand, this means that denotational order is always a sufficient condition for refinement, justifying its traditional use as refinement relation in observation-oriented semantics. Vice versa, it shows that testing preorder is a potentially weaker notion of refinement than denotational order, as it should be, being a formalization of the effects of the observational constraints imposed by $\mathcal{L}$'s operators.

An interesting question is whether testing preorder fully covers the equivalences induced by these observability constraints, or whether the same testing procedure, applied to the testing preorder instead of the denotational order, would again weaken the relation. Fortunately this is not the case, as the following theorem shows.

Assume that the success criteria $s \in S$ are expressible by $\mathcal{L}$-programs, i.e. that for each $s \in S$ there is an $\mathcal{L}$-program π_s with $s = \mathcal{I}[\![\pi_s]\!]$. This can always be achieved by extending $\mathcal{L}$ to $\mathcal{L}' \stackrel{\text{def}}{=} (T, Ax, O \uplus S)$, where each $s \in S$ is an 0-ary operator of sort $\langle\rangle \to Ax$, interpreted by the $\mathcal{L}'$-semantics $\mathcal{I}'$ which is $\mathcal{I}$ extended by the identity mapping for the operators $s \in S$, i.e. each $s \in S$ is interpreted by itself.

Furthermore we define a family $(\leq_i)_{i \in \mathbb{N}}$ of relations on $\mathcal{L}$-terms by, for each sort t and all $\mathcal{L}$-terms P and Q of sort t,

$$\begin{array}{lll} P \leq_0 Q & \text{iff} & \mathcal{I}[\![P]\!] \sqsubseteq_{\mathcal{I}_t} \mathcal{I}[\![Q]\!] \ , \text{ and} \\ P \leq_{i+1} Q & \text{iff} & s \leq_i c(P) \ \Rightarrow \ s \leq_i c(Q) \quad \text{for each } s \in S \text{ and each } t\text{-context } c. \end{array}$$

Theorem 10.7 (Test abstraction is idempotent). *$(\leq_i) = (\leq_j)$ for each $i, j \geq 1$, i.e. coarsening the refinement order by testing is idempotent, stabilizing after just one application.*

Because of the inductive definition of $\leq_i$ this follows immediately from

Lemma 10.8. $(\leq_1) = (\leq_2)$.

Proof. First we show $(\leq_1) \subseteq (\leq_2)$. Theorem 10.4 shows that all operators in O are monotonic with respect to $\leq_1$. Hence, an argument completely analogous to the proof of theorem 10.6 shows $P \leq_1 Q \Rightarrow P \leq_2 Q$.

To prove the reverse inclusion we have to show that $P \leq_2 Q$ implies $P \leq_1 Q$. Therefore, let P and Q be $\mathcal{L}$-terms of sort t such that $P \leq_2 Q$ and let c be a t-context with $s \leq_0 c(P)$. Then by definition of $\leq_1$ we have to show $s \leq_0 c(Q)$.

By definition of $\leq_2$, $P \leq_2 Q$ means

$$s_1 \leq_1 c_1(P) \ \Rightarrow \ s_1 \leq_1 c_1(Q) \quad \text{for each } t\text{-context } c_1 \text{ and each } s_1 \in S$$

which by definition of $\leq_1$ is equivalent to $(\alpha) \Rightarrow (\beta)$ for each $s_0 \in S$ and each Ax-context c_0, with

$$\begin{array}{llll} (\alpha) & s_0 \leq_0 c_0(s_1) & \Rightarrow & s_0 \leq_0 c_0(c_1(P)) \quad \text{and} \\ (\beta) & s_0 \leq_0 c_0(s_1) & \Rightarrow & s_0 \leq_0 c_0(c_1(Q)) \ . \end{array}$$

Now, taking $s_0 \stackrel{\text{def}}{=} s$ and $c_1 \stackrel{\text{def}}{=} c$ and $c_0 \stackrel{\text{def}}{=} \sqcup$ (the empty Ax-context) we have by $s \leq_0 c(P)$ that $s_0 = s \leq_0 c(P) = \sqcup(c(P)) = c_0(c_1(P))$. Hence, (α) holds, which by $P \leq_2 Q$ implies (β), i.e.

$$s \leq_0 s_1 \;\Rightarrow\; s \leq_0 c(Q)$$

by the specific choice of s_0, c_0, and c_1. Now, taking $s_1 \stackrel{\text{def}}{=} s$, reflexivity of $\leq_0$ implies $s \leq_0 c(Q)$ as desired. ▟

This theorem shows that the test preorder obtained from a given semantics is unique, allowing us to speak of *the* test preorder associated to a given semantics and a given set of success criteria. Therefore, we will in the remainder just write $\leq$ instead of $\leq_i$ for the testing preorder.

Now that we have made clear that testing preorders

- describe our intuition that replacement of a component is — without further checking the context — only admissible if the overall system cannot loose desirable properties due to that replacement,
- may indeed provide additional implementation freedom (cf. theorem 10.6), and
- provide a closure operation on refinement relations (theorem 10.7)

the question arises how to put their apparent advantages to use. Basically, there are three such possibilities:

1. *To explicitly construct the testing preorder, i.e. to provide a fully abstract semantics.*

As is well-known from the literature, this goal is often very hard to achieve. Given the complete lack of fully abstract semantics for languages dealing with metric real-time, this approach seems hardly practicable in the field of applications we are dealing with, at least with the level of maturity that this field has currently reached. Therefore, we concentrate on the other options:

2. *To equip a single language $\mathcal{L}$ with two (or even more) different semantics which give rise to the same testing preorder.*

   If both denotational orders give rise to the same closure under testing, theorem 10.6 implies that they are both subrelations of the testing preorder induced. Hence, both order relations may be freely mixed in refinement steps during system development without risking invalidation of any desirable system property. This allows to select the more advantageous semantic model independently for each individual refinement step.

3. *To accept more liberal refinement relations than the denotational order whenever they are known to be subrelations of the testing preorder.*

   The advantage is that generally there is no need to explicitly define the full testing preorder, which would amount to building a fully abstract model, but just sufficiently large relations contained in it.

In the remainder of this chapter we will be mostly concerned with a proof rule for a relation being a subrelation of the testing preorder, making 3. workable.

10.3 Testing-adequate refinement relations

In general it is very difficult, if not even impracticable, to provide an explicit definition of the full testing preorder. Hence, although an adequate refinement relation, it has in general not attracted much attention in the field of real-time concurrency.

We conjecture that it is in general not necessary to construct the testing preorder explicitly for taking advantage of its liberalized notion of refinement: it is perfectly convenient to take whatever refinement relation seems to be adequate, just making sure that it is contained in the testing preorder such that refinement is compatible with the testing preorder. It is the objective of this section to provide a *simple* proof rule for the latter fact.

The common property of all relations that may be safely used as refinement relations is that they are subrelations of the testing preorder.

Definition 10.9 (Testing-adequacy). *A relation I on $\mathcal{L}$-terms is called* testing-adequate *(with respect to semantics $\mathcal{I}$ and success criteria S) iff $I \subseteq (\leq)$.*

Any such relation may be used for refining an arbitrary component of a system without risking invalidation of a success criterion, i.e. of an important demand on the overall system.

By theorem 10.6 the denotational order is testing-adequate. But also larger relations, which more versatile in refinement, can easily be shown to be testing-adequate:

Theorem 10.10 (Proof rule for testing-adequacy). *Suppose I is a relation on $\mathcal{L}$-terms that satisfies*

(i) if $(P, Q) \in I$ then P and Q have the same sort,

(ii) for each pair (π, ψ) of $\mathcal{L}$-programs with $(\pi, \psi) \in I$ and each $s \in S$ the implication $s \sqsubseteq \mathcal{I}[\![\pi]\!] \Rightarrow s \sqsubseteq \mathcal{I}[\![\psi]\!]$ holds, and

(iii) for each operator $o : t_1, \ldots, t_n \to t$ in O and all $\mathcal{L}$-terms $P_1, \ldots, P_n$ and $Q_1, \ldots, Q_n$ of sorts $t_1, \ldots, t_n$ the implication

$$(\forall i \in \{1, \ldots, n\} \,.\, (P_i, Q_i) \in I\,) \Rightarrow (o(P_1, \ldots, P_n), o(Q_1, \ldots, Q_n)) \in I$$

holds.

Then I is testing-adequate with respect to $\mathcal{I}$ and S.

Proof. Let $(P, Q) \in I$. By (i), P and Q are $\mathcal{L}$-terms of a common sort t. Induction over (iii) shows that $(P, Q) \in I$ implies $(c(P), c(Q)) \in I$ for arbitrary t-contexts c. By (ii) this implies $s \sqsubseteq \mathcal{I}[\![c(P)]\!] \Rightarrow s \sqsubseteq \mathcal{I}[\![c(Q)]\!]$ for arbitrary $s \in S$ and arbitrary t-contexts c, i.e. $P \leq Q$. ∎

Hence, for an arbitrarily defined relation on $\mathcal{L}$-terms there is a simple proof rule for showing that it is an admissible refinement relation: it suffices to show that it is a congruence for $\mathcal{L}$'s operators and that it preserves success conditions on $\mathcal{L}$-programs. Note that such a relation is similar to an algebraic relation, as suggested by G. Plotkin and S. Abramsky for analyzing applicative structures [Mit90, p. 428], with the base-relation on systems additionally being confined to provide the desired refinement property.

The economy about the proof rule of theorem 10.10 is that we did not *require* the refinement relation to be the full testing preorder, giving freedom of choice of the relation and a simple proof rule for its admissibility. On the other hand, the simplicity of the proof rule of theorem 10.10 could prevent us from really taking advantage of the more liberal notion of refinement provided by the testing preorder, as conditions (*i*) to (*iii*) may not hold for a sufficiently rich class of relations. Fortunately this is not the case, as the following theorem and its corollary show.

Theorem 10.11. *Let I be a relation on $\mathcal{L}$-terms satisfying (iii). Then I is testing-adequate if and only if (i) and (ii) hold.*

Proof. For the implication not covered by theorem 10.10 just note that $\leq$ is by definition a relation only relating $\mathcal{L}$-terms of same sort, i.e. (*i*) holds for every subrelation of $\leq$, and that due to the existence of the empty Ax-context $\sqcup$, $\mathcal{L}$-programs π and ψ with $\pi \leq \psi$ satisfy $s \sqsubseteq \mathcal{I}[\![\pi]\!] \Rightarrow s \sqsubseteq \mathcal{I}[\![\psi]\!]$. Hence every subrelation of $\leq$, i.e. any testing-adequate relation, satisfies (*ii*). ▟

As the testing preorder $\leq$ is trivially testing-adequate and satisfies (*iii*) by theorem 10.4, a direct consequence is

Corollary 10.12. $\leq$ *satisfies (i), (ii), and (iii).*

Hence the proof rule of theorem 10.10 is general enough to admit even the full testing preorder, allowing us to take advantage of it whenever we manage to construct it.

10.4 Compiling correctness

As the central property of our notion of correctness is preservation of success criteria on *complete* systems we can also apply it to implementation processes where a change of technology, internal interfaces, etc., is involved like in compilation. The peculiarity about compiler correctness is that a system during compilation is re-implemented in a different language, normally having completely different interfaces between components, different semantics, etc., such that the two levels cannot directly be related. But clearly, at least in the field of embedded control systems, this can only hold for internal interfaces of the control system. At the borders of the control system, interfaces of and demands on the control system cannot change during development, as we cannot simply redefine the physical world that we connect the controller to.[4] Hence, at the syntactically outermost level (that of *complete*

[4]Clearly, our semantic understanding does not directly relate to the physical environment, but only to models thereof, where some freedom of choice is left between different models. Nevertheless, that should not

control systems directly interfacing to the physical environment), the semantics of source and target have to have a common semantic domain as they interface in an equivalent way to the same (model of) physical environments.

As a consequence, the demands that may be placed upon a control system can be homogeneously understood at both the source and the target level. This makes preservation of the demands satisfied by a complete system also a reasonable notion of implementation across the source-to-target boundary. In particular, internal restructuring is permitted iff no external condition may be invalidated thereby. E.g., in the control architecture depicted in figure 2.1 on page 12, demands are originally concerned with the interface to the controlled physical plant, i.e. with the interface where sensors and actuators connect to it, and not with the interface between control system and sensors or actuators or even with internal interfaces of the control system. Hence, all interfaces of the control system, be it internal interfaces or those to sensors or actuators, may be freely altered if this is done sufficiently homogeneous.

To formalize this, let $\mathcal{L} = (T, Ax, O)$ and $\tilde{\mathcal{L}} = (\tilde{T}, \widetilde{Ax}, \tilde{O})$ be languages and let $\mathcal{I}$ and $\tilde{\mathcal{I}}$ be $\mathcal{L}$- and $\tilde{\mathcal{L}}$-semantics with a common semantic domain $\mathcal{I}_{Ax} = \tilde{\mathcal{I}}_{\widetilde{Ax}}$ for programs. Let $S \subseteq D_{\mathcal{I}_{Ax}}$ be a set of success criteria.

Definition 10.13 (Implementation correctness). *An $\tilde{\mathcal{L}}$-program ψ* implements *an $\mathcal{L}$-program π iff π $\mathtt{sat}$ $s \Rightarrow \psi$ $\mathtt{sat}$ s, i.e. $s \sqsubseteq \mathcal{I}[\![\pi]\!] \Rightarrow s \sqsubseteq \tilde{\mathcal{I}}[\![\psi]\!]$, holds for each $s \in S$.*

So far, the notion of implementation relates only complete systems. As a consequence a large amount of implementation freedom is permitted for internal structures as long as the complete systems meet the implementation criterion. On the other hand, the implementation correctness criterion for complete systems is quite rigorous, formalizing the idea of an implementation meeting all the demands on the original system. Now, let C be a relation between $\mathcal{L}$-terms and $\tilde{\mathcal{L}}$-terms.

Definition 10.14 (Compiler correctness). *C is a* correct compiler *if and only if it relates $\mathcal{L}$-programs to implementations thereof, i.e. iff $C \cap (\mathcal{L}Prog \times \tilde{\mathcal{L}}Prog) \subseteq Imp$ for $Imp \stackrel{\text{def}}{=} \{(\pi, \psi) \in \mathcal{L}Prog \times \tilde{\mathcal{L}}Prog \mid \psi$ implements $\pi\}$, where $\mathcal{L}Prog$ denotes the set of $\mathcal{L}$-programs and $\tilde{\mathcal{L}}Prog$ the set of $\tilde{\mathcal{L}}$-programs, respectively.*

For effective compilation we will normally be interested in relations from $\mathcal{L}$-terms to $\tilde{\mathcal{L}}$-terms which are mediated by a finite set of structural rules. I.e., in compilation one is not interested in arbitrary relations between source and target programs, not even in arbitrary computable relations, but in relations that obey a structurally inductive definition.

Definition 10.15 (Structurally defined relation). *A* structurally defined relation *is a relation between terms that can be defined via structural induction over its first argument: let*

- *$C_{\text{type}} : T \to \tilde{T}$ be a total mapping from $\mathcal{L}$-types to $\tilde{\mathcal{L}}$-types,*

invalidate our argument, as the system — to be safe — should not depend on the peculiarities of a specific model.

- $C_{\text{base}} \subseteq \bigcup_{t \in T} \left(\mathcal{L}_t \times \tilde{\mathcal{L}}_{C_{\text{type}}(t)}\right)$, *and*

- $C_{\text{struct}} \subseteq \left\{ (o,e) \;\middle|\; \begin{array}{l} o \in O, \\ (o : t_1, \ldots, t_n \to t) \Rightarrow \left(e \in \tilde{\mathcal{L}}^{\sqcup_1,\ldots,\sqcup_n}_{C_{\text{type}}(t)}\right) \end{array} \right\}$,

  where $\tilde{\mathcal{L}}^{\sqcup_1,\ldots,\sqcup_n}_{C_{\text{type}}(t)}$ *is the set of terms of type* $C_{\text{type}}(t)$ *of the language* $\tilde{\mathcal{L}}$ *extended with placeholders* $\sqcup_1, \ldots, \sqcup_n$ *of types* $\langle\rangle \to C_{\text{type}}(t_i)$.

The structural relation *defined by* $(C_{\text{type}}, C_{\text{base}}, C_{\text{struct}})$ *is the smallest relation that satisfies*

1. $C \supseteq C_{\text{base}}$, *and*

2. *if* $o : t_1, \ldots, t_n \to t$ *and* $P_1, \ldots, P_n$ *are* $\mathcal{L}$*-terms of types* $t_1, \ldots, t_n$ *and* $(P_i, Q_i) \in C$ *for each* $i \in \{1, \ldots, n\}$ *then*

$$(o(P_1, \ldots, P_n), e(Q_1, \ldots, Q_n)) \in C \quad ,$$

   where $e(Q_1, \ldots, Q_n)$ *denotes* e *with* $\sqcup_i$ *being substituted by* Q_i.

If C *is the structural relation induced by* $(C_{\text{type}}, C_{\text{base}}, C_{\text{struct}})$ *then we call* C_{base} *its* base relation *and* C_{struct} *its* structural rules.

Note that we did not require the base relation to be concerned with atomic $\mathcal{L}$-terms only. This generality is sometimes needed in practice. Unfortunately, it has also the pathological consequence that in principle any relation C between $\mathcal{L}$- and $\tilde{\mathcal{L}}$-*programs* can be given a structural definition by defining $C_{\text{base}} \stackrel{\text{def}}{=} C$ and $C_{\text{struct}} \stackrel{\text{def}}{=} \emptyset$. However, in practical applications, both the base relation C_{base} and the structural rules C_{struct} have to be finite, and the potential of compiling infinitely many source programs comes from iterated application of the structural rule. It is this particular case that our compiler verification techniques will be tailored towards, although they remain sound in the pathological cases also.

In the remainder, let the compiling relation C be the structural relation induced by $(C_{\text{type}}, C_{\text{base}}, C_{\text{struct}})$. For an arbitrary $t \in T$ we can easily extend C to a relation C^t between $\mathcal{L}^t$ (the language of t-contexts of $\mathcal{L}$) and $\tilde{\mathcal{L}}^{C_{\text{type}}(t)}$ (the language of $C_{\text{type}}(t)$-contexts of $\tilde{\mathcal{L}}$) by mapping $\mathcal{L}^t$'s operator $\sqcup$ (the place-holder of type t) to $\tilde{\mathcal{L}}^{C_{\text{type}}(t)}$'s operator $\sqcup$ (the place-holder of type $C_{\text{type}}(t)$) in the base relation. I.e., C^t is the relation that is structurally induced by $(C_{\text{type}}, C_{\text{base}} \cup \{\sqcup, \sqcup\}, C_{\text{struct}})$. Then we can define

Definition 10.16 (Refinement modulo an implementation relation). *Let* $t \in T$, P *an* $\mathcal{L}$*-term of type* t, Q *an* $\tilde{\mathcal{L}}$*-term of type* $C_{\text{type}}(t)$. *Then* P *is refined by* Q *modulo* C, *denoted* $P \leq_C Q$, *iff* $s \sqsubseteq \mathcal{I}[\![c(P)]\!] \Rightarrow s \sqsubseteq \tilde{\mathcal{I}}[\![\tilde{c}(Q)]\!]$ *holds for each* t*-context* c *and each* $C_{\text{type}}(t)$ *context* $\tilde{c}$ *with* $(c, \tilde{c}) \in C^t$ *and each* $s \in S$.

The reader should not be worried by the fact that this notion of refinement depends on the implementation relation whereas we are used to think of the refinement relation as an absolute item with respect to which we establish and verify the implementation. This mutual dependency is a direct consequence of the insight that the internal interaction between components is irrelevant if the external demands are preserved and hence it is part of the enrichment of the refinement relation: indeed, the mutual dependency between the compilation relation C and the induced refinement relation $\leq_C$ is closely related to compiler correctness.

Lemma 10.17 (Dependency between C and $\leq_C$, and compiler correctness). *$C \cap (\mathcal{L}Prog \times \tilde{\mathcal{L}}Prog) \subseteq Imp$ iff $C \subseteq (\leq_C)$, i.e. C is a correct compiler if and only if it is a subset of its induced refinement relation.*

Proof. We start by showing that $(C \cap (\mathcal{L}Prog \times \tilde{\mathcal{L}}Prog)) \subseteq Imp$ implies $C \subseteq (\leq_C)$. Therefore assume that $(C \cap (\mathcal{L}Prog \times \tilde{\mathcal{L}}Prog)) \subseteq Imp$ holds and that P is an $\mathcal{L}$-term of type $t \in T$ and Q an arbitrary $\tilde{\mathcal{L}}$-term with $(P, Q) \in C$.

Let c be an arbitrary t-context (in $\mathcal{L}$) and $\tilde{c}$ an arbitrary $C_{\text{type}}(t)$-context (in $\tilde{\mathcal{L}}$) with $(c, \tilde{c}) \in C^t$. Due to the structural definition of C it follows that $(c(P), \tilde{c}(Q)) \in C$ and hence, as $(C \cap (\mathcal{L}Prog \times \tilde{\mathcal{L}}Prog)) \subseteq Imp$, $s \sqsubseteq \mathcal{I}[\![c(P)]\!] \Rightarrow s \sqsubseteq \tilde{\mathcal{I}}[\![\tilde{c}(Q)]\!]$ for each $s \in S$. I.e., as $(c, \tilde{c}) \in C^t$ was arbitrarily chosen, $P \leq_C Q$ and thus $C \subseteq (\leq_C)$.

To show the other implication we assume that $C \subseteq (\leq_C)$ and that π and ψ are arbitrary $\mathcal{L}$- and $\tilde{\mathcal{L}}$-programs with $(\pi, \psi) \in C$. Then by $C \subseteq (\leq_C)$ and the definition of $\leq_C$ we have

$$s \sqsubseteq \mathcal{I}[\![c(\pi)]\!] \Rightarrow s \sqsubseteq \tilde{\mathcal{I}}[\![\tilde{c}(\psi)]\!] \text{ for each } (c, \tilde{c}) \in C^{Ax} \text{ and each } s \in S. \tag{10.2}$$

By definition, C^{Ax} contains at least the pair $(\sqcup, \sqcup)$ consisting of the empty Ax-contexts of $\mathcal{L}$ and $\tilde{\mathcal{L}}$. As $\sqcup(\pi) = \pi$ and $\sqcup(\psi) = \psi$ we can deduce from (10.2) that

$$s \sqsubseteq \mathcal{I}[\![\pi]\!] \Rightarrow s \sqsubseteq \tilde{\mathcal{I}}[\![\psi]\!] \text{ for each } s \in S.$$

I.e. ψ implements π and hence $(\pi, \psi) \in Imp$. As π and ψ are arbitrarily chosen from $C \cap (\mathcal{L}Prog \times \tilde{\mathcal{L}}Prog)$ this shows $(C \cap (\mathcal{L}Prog \times \tilde{\mathcal{L}}Prog)) \subseteq Imp$. ▟

As $(C \cap (\mathcal{L}Prog \times \tilde{\mathcal{L}}Prog)) \subseteq Imp$ is the natural condition for compiler correctness, lemma 10.17 states that a compositional relation C between $\mathcal{L}$ and $\tilde{\mathcal{L}}$ is a correct compiler *if and only if* $C \subseteq (\leq_C)$. In order to be able to establish the latter inclusion as a means of compiler verification we adapt our proof rule for testing-adequacy from theorem 10.10.

Lemma 10.18. *Suppose I is a relation between $\mathcal{L}$ and $\tilde{\mathcal{L}}$-terms that satisfies*

(i) if $(P, Q) \in I$ and P has type t then Q has type $C_{\text{type}}(t)$,

(ii) for each $\mathcal{L}$-program π and each $\tilde{\mathcal{L}}$-program ψ with $(\pi, \psi) \in I$ it holds that $s \sqsubseteq \mathcal{I}[\![\pi]\!] \Rightarrow s \sqsubseteq \tilde{\mathcal{I}}[\![\psi]\!]$ for each $s \in S$,

(iii) for each operator $o : t_1, \ldots, t_n \to t$ in O and all $\mathcal{L}$-terms $P_1, \ldots, P_n$ of types $t_1, \ldots, t_n$ and all $\tilde{\mathcal{L}}$-terms $Q_1, \ldots, Q_n$ of types $C_{\text{type}}(t_1), \ldots, C_{\text{type}}(t_n)$ the implication

$$(\forall i \in \{1, \ldots, n\} \,.\, (P_i, Q_i) \in I\,) \Rightarrow (o(P_1, \ldots, P_n), e(Q_1, \ldots, Q_n)) \in I$$

holds for each $(o, e) \in C_{\text{struct}}$.

Then $I \subseteq (\leq_C)$ holds.

Proof. Similar to the proof of theorem 10.10. ∎

Furthermore, under the conditions of lemma 10.18 and the additional condition

(*iv*) $C_{\text{base}} \subseteq I$,

the structural relation C induced by $(C_{\text{type}}, C_{\text{base}}, C_{\text{struct}})$ is obviously confined to be a subrelation of I:

Lemma 10.19. *If I is a relation between $\mathcal{L}$- and $\tilde{\mathcal{L}}$-terms satisfying (iii) and (iv) then $C \subseteq I$.*

Proof. By structural induction on $\mathcal{L}$-terms. ∎

Lemmata 10.18 and 10.19 show that a relation I satisfying conditions (*i*) to (*iv*) contains C and is itself contained in $\leq_C$. Hence

Corollary 10.20. *If I is a relation between $\mathcal{L}$- and $\tilde{\mathcal{L}}$-terms satisfying (i), (ii), (iii), and (iv) then $C \subseteq (\leq_C)$.* ∎

By lemma 10.17, this implies

Theorem 10.21 (Compiler verification). *If I is a relation between $\mathcal{L}$- and $\tilde{\mathcal{L}}$-terms satisfying (i), (ii), (iii), and (iv) then C is a correct compiler.* ∎

Hence, conditions (*i*) to (*iv*) of lemma 10.18 provide a proof rule for correctness of compositional compilers. It supports even an incremental technique of defining and verifying the compiling relation C, where — instead of doing a one-shot definition of the whole compiler — only a single operator is dealt with at a time. Such an approach has been advocated by C.A.R. Hoare in [Hoa90], using an a-priori fixed refinement relation. To do the same in our more flexible setting, one has to define the implicit implementation relation I first (e.g. by defining a simulation) and to show that it satisfies (*i*) and (*ii*). Then one can start to incrementally define C operator-by-operator, thereby modularly showing (*iii*) or (*iv*), depending on whether a structural rule or a base case is defined.

10.5 Discussion

In this chapter we have shown how a rigorous concept of refinement can be established that is more liberal than trajectory inclusion, yet applicable in contexts where interaction with physical environments has to be maintained. The notion of refinement proposed accepts a system as a refinement of another iff there is no application context in which the refining system can invalidate an expectation that is met by the refined system. As it is in general difficult to define a semantics that directly supports this notion of refinement, proof rules supporting it on a non-dedicated, e.g. observation-oriented, semantics have been given. These proof rules take the form of simple inductive arguments verifying the admissibility of relations as refinement relations or even as compilers.

Natural fields of application for this theory are classical engineering examples of interface refinements that go well beyond the scope of the trajectory-based model of refinement. Examples of such interface refinements are VLSI implementations of digital circuitry as they implement systems originally described as having discrete state and discrete communication alphabet by analogue devices communicating continuous signals. In chapter 12, we will provide a rigorous proof of such an implementation.

11

Hybrid Duration Logic

Duration Logic, as defined in chapter 5, is a logic tailored towards reasoning about durational constraints on time-dependent Boolean-valued states. This provides a very expressive metric dense-time temporal logic. Nonetheless, it is inherently limited to Boolean state or at least, using appropriate encodings, discrete state. While this is a reasonable restriction when dealing with discrete systems only, e.g. when reasoning within digital models of hardware behaviour, it precludes dealing with continuous components within the same formalism. Such an integration of discrete and continuous reasoning within a single formalism has recently attracted much interest as its prospects for analyzing interactions of discrete and continuous systems seem promising. Thus, such combined formalisms, e.g. those proposed in [MP93b, HMP93, Lam93, NOSY93, ACHH93, NK93, ZRH93, Str94, NT94], may help in developing systems consisting of a non-trivial mixture of discrete and continuous components, so-called *hybrid systems*. The hope is that using a single formalism for dealing with both discrete and continuous aspects of hybrid systems might allow to freely mix discrete and continuous analysis techniques, thereby combining their virtues.

The hybrid system formalisms proposed so far have been derived by extending non-hybrid ones, either through adding discrete components to well-established control-theoretic notation [Str94], or by adding continuous variables and differential or integral equations to discrete formalisms [MP93b, HMP93, Lam93, NOSY93, ACHH93, NK93, ZRH93, NT94]. In this chapter we will study a similar extension of Duration Logic, albeit for quite different reasons: we are not interested in designing hybrid systems, but in analyzing how some continuous devices, namely networks of transistors, implement discrete systems. For bridging this semantic gap we also need a means of combining discrete and continuous analysis techniques. The technicalities involved are much the same as those involved in extending Duration Calculus towards a design calculus for hybrid systems. In this chapter we take advantage of this analogy by following the lines of [ZRH92], a precursor of [ZRH93], in extending DL towards a hybrid logic.

11.1 Syntactic extensions

The main concept in our extension of Duration Logic towards a formalism incorporating continuous analysis are integral equations and inequations. Five major additions to DL suffice to yield Hybrid Duration Logic HDL:

1. piecewise continuous variables and functions thereof,
2. integrals over time,
3. initial and final values of piecewise continuous variables or functions thereof within an observation interval,
4. arithmetic expressions built from integrals and initial and final values, and
5. equations and inequations built from those integral expressions.

Furthermore, for convenience of reasoning in HDL, we will add existential quantification of time-dependent states.

11.1.1 Piecewise continuous state variables and expressions

In order to be able to speak of both piecewise continuous and discrete time-dependent states, we introduce a second class of state variables, to be interpreted by piecewise continuous functions of time:

$$\langle \mathit{continuous\,state\,variable} \rangle \quad ::\in \quad \mathit{CVarname} \ ,$$

where the set *CVarname* of names of piecewise continuous state variables is disjoint from the set *Varname* of discrete state variables. In the remainder, we will decorate elements of *CVarname* with a tilde, whereas elements of *Varname* remain undecorated such that $\tilde{x}, \tilde{y}, \ldots, \tilde{x}', \tilde{y}', \ldots$ are piecewise continuous state variables whereas $x, y, \ldots, x', y', \ldots$ are discrete ones.

Semantically, piecewise continuous state variables will be interpreted by total, piecewise continuous, Riemann-integrable real-valued functions of *Time*. I.e., their interpretations will be continuous over time except for finitely many discontinuity points within any finite interval of *Time*, and Riemann-integrable over any finite interval of *Time*. Unfortunately, this class of functions is not closed under some common arithmetic operations like division. Nevertheless, we allow them to be composed by all the usual arithmetical operations.

$$\begin{array}{rcl}\langle \mathit{state\,expression} \rangle & ::= & \langle \mathit{continuous\,state\,variable} \rangle \mid \\ & & \langle \mathit{constant} \rangle \mid \\ & & -\langle \mathit{state\,expression} \rangle \mid \\ & & \langle \mathit{state\,expression} \rangle + \langle \mathit{state\,expression} \rangle \mid \\ & & \langle \mathit{state\,expression} \rangle - \langle \mathit{state\,expression} \rangle \mid \\ & & \langle \mathit{state\,expression} \rangle \cdot \langle \mathit{state\,expression} \rangle \mid \\ & & \langle \mathit{state\,expression} \rangle / \langle \mathit{state\,expression} \rangle \mid \\ & & \ldots \end{array}$$

Whenever such a state expression may give rise to a function of *Time* that is not Riemann-integrable within the current observation interval then this is to be interpreted as a side-condition that the involved variables have to be interpreted by functions that do not give rise to undefined integral or border-value expressions. We will expand on this in section 11.2 when we define the semantics.

11.1.2 Integrals over time

As state expressions are interpreted by Riemann-integrable functions of *Time*, it is straightforward to introduce integration over time. The integral operator has no explicit bounds associated, instead it yields the integral of the enclosed state expression over the current observation interval.

$$\langle \mathit{state\,integral} \rangle \quad ::= \quad \int\langle \mathit{state\,expression} \rangle$$

11.1.3 Initial and final values

Initial and final values of state expressions in an observation interval are often convenient for expressing boundary conditions of an integral equation or inequation. We offer them through two operators `b.` and `e.` yielding the limit from the right in the left endpoint and the limit from the left in the right endpoint of the current observation interval.

$$\begin{array}{lcl} \langle \mathit{border\ value} \rangle & ::= & \mathtt{b.}\langle \mathit{state\,expression} \rangle \mid \\ & & \mathtt{e.}\langle \mathit{state\,expression} \rangle \end{array}$$

Note that these limits need not necessarily exist.

11.1.4 Integral and boundary value expressions

Integrals over time and initial and final value expressions can be combined using arithmetic operators to form more complex expressions.

$$\begin{array}{lcl} \langle \mathit{integral\,expression} \rangle & ::= & \langle \mathit{state\,integral} \rangle \mid \\ & & \langle \mathit{border\ value} \rangle \mid \\ & & \langle \mathit{constant} \rangle \mid \\ & & -\langle \mathit{integral\,expression} \rangle \mid \\ & & \langle \mathit{integral\,expression} \rangle + \langle \mathit{integral\,expression} \rangle \mid \\ & & \langle \mathit{integral\,expression} \rangle - \langle \mathit{integral\,expression} \rangle \mid \\ & & \langle \mathit{integral\,expression} \rangle \cdot \langle \mathit{integral\,expression} \rangle \mid \\ & & \langle \mathit{integral\,expression} \rangle / \langle \mathit{integral\,expression} \rangle \mid \\ & & \ldots \end{array}$$

Again, some operations like division may introduce undefinedness of an integral expression on some well-defined trajectory for some observation intervals. Similarly to state expressions, this is interpreted as a side-condition that only trajectories that do not give rise to such partiality of any state expression occurring in the formula under consideration have to be taken into account. Technically, this is done by defining that atomic formulae are not satisfied if a contained integral expression is undefined.

11.1.5 Integral equations and inequations

Instead of explicitly adding a syntactic category $\langle \mathit{integral\,inequation} \rangle$, we take advantage of the concepts already offered by DL and just extend the syntax of formulae through a new production for atomic formulae:

$$\langle atomic\,formula\rangle \quad ::= \quad \langle integral\,expression\rangle \geq 0$$

By integral expressions being closed under subtraction and formulae being closed under negation and conjunction, the familiar kinds of integral equations and inequations can be built from this. In particular, we define the abbreviations

$$\begin{aligned} e_1 \geq e_2 &\stackrel{\text{def}}{=} e_1 - e_2 \geq 0 \ , \\ e_1 \leq e_2 &\stackrel{\text{def}}{=} e_2 \geq e_1 \ , \\ e_1 = e_2 &\stackrel{\text{def}}{=} (e_1 \leq e_2) \wedge (e_1 \geq e_2) \end{aligned}$$

for building equations or inequations from integral expressions e_1 and e_2.

11.1.6 Existential quantification of time-dependent states

When describing the dynamic behaviour of hardware systems by formulae of HDL, a mechanism for hiding externally invisible internal detail is often desirable. Since the early days of predicative semantics [Heh84], existential quantification has proven to be an adequate logical means for this. Hence, we allow existential quantification of discrete as well as piecewise continuous time-dependent states in HDL.

$$\begin{aligned} \langle formula\rangle \quad ::= \quad & \exists \langle state\,variable\rangle \,.\, \langle formula\rangle \ \mid \\ & \exists \langle continuous\,state\,variable\rangle \,.\, \langle formula\rangle \end{aligned}$$

The straightforward interpretation is that an existentially quantified formula is satisfied on a finite observation iff there is a (time-dependent) substitution for that variable within that observation that satisfies the enclosed formula.

11.2 Semantics

Hybrid Duration Logic is interpreted over trajectories that give time-dependent, finitely variable, Boolean-valued interpretations to discrete state variables and time-dependent, piecewise continuous, Riemann-integrable, real-valued interpretations to continuous state variables.

$$tr \in CTraj \stackrel{\text{def}}{=} \left\{ tr : Time \to \Sigma \;\middle|\; \begin{array}{ll} t \mapsto tr(t)(v) & \text{finitely variable for each} \\ & v \in Varname, \\ t \mapsto tr(t)(\tilde{v}) & \text{piecewise continuous and} \\ & \text{Riemann-integrable for} \\ & \text{each } \tilde{v} \in CVarname \end{array} \right\} ,$$

where

$$\Sigma \stackrel{\text{def}}{=} \left\{ \sigma : (Varname \cup CVarname) \to (\mathbb{B} \cup \mathbb{R}) \;\middle|\; \begin{array}{ll} \sigma(v) \in \mathbb{B} & \text{for each} \\ & v \in Varname, \\ \sigma(\tilde{v}) \in \mathbb{R} & \text{for each} \\ & \tilde{v} \in CVarname \end{array} \right\} .$$

As in DL, satisfaction of an HDL formula ϕ by a trajectory tr is defined as a limit property over the chain $\{(tr, [0, e]) \mid e \in \mathit{Time}\}$ of prefix-observations of tr. Satisfaction of HDL formulae by finite observations is defined similarly to DL.

11.2.1 Satisfaction by finite observations

Indeed, just as DL is a syntactic subset of HDL, the semantics of HDL is defined to coincide on DL with the old semantics. All the definitions of section 5.2 remain valid, with the only exception that the trajectory set *Traj* has to be replaced by the new set *CTraj*. Hence, we can concentrate on the interpretation of the new constructs.

In the remainder let $(tr, [a, b]) \in \mathit{Obs}_{\mathit{CTraj}} = \mathit{CTraj} \times \mathit{TimeInterval}$ be an observation, i.e. a pair of a time-dependent valuation of all state variables and a finite observation interval.

Semantics of state expressions

Just like state assertions, state expressions are to be evaluated in time instants. Hence, their semantics depends on a state. This yields a mapping $[\![\cdot]\!] : \mathit{StateExpressions} \rightarrow \Sigma \xrightarrow{\text{partial}} \mathbb{R}$ defined by

$$\begin{aligned}
[\![\tilde{v}]\!](\sigma) &\stackrel{\text{def}}{=} \sigma(\tilde{v}) && \text{for each state variable } \tilde{v} \in \mathit{CVarname}, \\
[\![k]\!](\sigma) &\stackrel{\text{def}}{=} k && \text{for each constant } k, \\
[\![-e]\!](\sigma) &\stackrel{\text{def}}{=} -[\![e]\!](\sigma) \ , \\
[\![e_1 + e_2]\!](\sigma) &\stackrel{\text{def}}{=} [\![e_1]\!](\sigma) + [\![e_2]\!](\sigma) \ ,
\end{aligned}$$

and similarly for the other arithmetic operators. Note that this mapping may become partial if partial arithmetic operators are involved.

Integral and boundary value expressions

Integral and boundary value expressions are interpreted on observations, yielding a real value if defined. I.e., the semantics of such expressions is a partial mapping from $\mathit{Obs}_{\mathit{CTraj}}$ to $\mathbb{R}$. For the base cases, namely integrals and initial and final values, it will be defined in the next two paragraphs. The extension to arbitrary arithmetic expressions built from those is straightforward.

Integrals. Integrals of state expressions are interpreted over observations, yielding the Riemann integral over the enclosed state expression along the observation interval. Given an observation $(tr, [a, b])$, the semantics of the state integral $\int e$ on that observation is

$$[\![\textstyle\int e]\!](tr, [a, b]) \stackrel{\text{def}}{=} \int_a^b ([\![e]\!] \circ tr)\,(t)\,\mathrm{d}t \ .$$

However, the Riemann-integral of a state expression need not always exist, even if all the contained state variables are interpreted by Riemann-integrable functions of time. Hence, the semantics of integrals of state expressions is a *partial* mapping from observations to $\mathbb{R}$. When defining satisfaction of a formula by an observation, avoiding undefinedness stemming from such partiality will become a side-condition on satisfaction.

Initial and final values. The two operators $\mathtt{b}.$ and $\mathtt{e}.$ yield the limit from the right in the left endpoint and the limit from the left in the right endpoint of the current observation interval.

$$\begin{aligned} [\![\mathtt{b}.e]\!](tr,[a,b]) &\stackrel{\text{def}}{=} \lim_{t>a} ([\![e]\!] \circ tr)(t) \ , \\ [\![\mathtt{e}.e]\!](tr,[a,b]) &\stackrel{\text{def}}{=} \lim_{t<b} ([\![e]\!] \circ tr)(t) \ . \end{aligned}$$

Again, these limits need not exist even though all state variables are interpreted by piecewise continuous functions of time. I.e., the semantics of $\mathtt{b}.e$ and $\mathtt{e}.e$ may be partial. Again, this will be taken care of by a side-condition on satisfaction of a formula by an observation.

Integral inequations

An integral inequation is an atomic formula of the form $e \geq 0$, where e is an integral expression. It is satisfied on an observation $(tr,[a,b]) \in Obs_{CTraj}$, denoted $tr,[a,b] \models e \geq 0$, iff the enclosed integral expression e is defined and non-negative on that observation:

$$tr,[a,b] \models e \geq 0 \quad \stackrel{\text{def}}{\iff} \quad \left(\begin{array}{l} [\![e]\!](tr,[a,b]) \text{ is defined } \wedge \\ [\![e]\!](tr,[a,b]) \geq 0 \end{array} \right) \ .$$

Note that $e \geq 0$ fails on an observation whenever e is undefined on that observation. By the semantics of negation this implies that, in contrast, its negation $\neg(e \geq 0)$ is trivially satisfied on observations which leave e undefined. This asymmetry is certainly not desirable.

At first glance, this might seem to necessitate a three-valued extension of the satisfaction relation. However, there is a simpler cure. Above semantics allows to define a definedness predicate

$$\mathrm{defd}(e) \stackrel{\text{def}}{=} e - e \geq 0 \ .$$

Using this, a predicate $e < 0$ that in analogy to $e \geq 0$ fails whenever e is undefined can be easily formalized:

$$e < 0 \stackrel{\text{def}}{=} \mathrm{defd}(e) \wedge \neg(e \geq 0) \ .$$

Formalizations of further inequations like $e_1 < e_2$ are then straightforward.

Due to the definedness predicate, a formal treatment of definedness is possible even in the two-valued logics HDL. However, undefinedness is hardly ever a desirable property in practice such that this expressiveness would mostly be used for formulating assumptions that undefinedness is not encountered. In our application of HDL in the next chapter we will not even face the problem.

Existential quantification

As usual, an existentially quantified formula is satisfied iff there is a possible substitution for the bound variable that makes the enclosed formula true. As existential quantification in HDL is on state variables, this substitution is on time-dependent functions rather than on base-values, giving a limited form of second-order quantification. For an arbitrary discrete or piecewise continuous variable v and an arbitrary HDL formula ϕ, satisfaction of $\exists\, v \,.\, \phi$ by the observation $(tr, [a, b])$ is defined as

$$tr, [a, b] \models \exists\, v \,.\, \phi \quad \stackrel{\text{def}}{\Longleftrightarrow} \quad \exists\, tr' \in \mathit{CTraj} \,. \left(\begin{array}{ll} tr', [a, b] \models \phi & \wedge \\ \forall\, w \in (\mathit{Varname} \cup \mathit{CVarname}) \setminus \{v\} \,. & \\ \quad \forall\, t \in \mathit{Time} \,.\, tr(t)(w) = tr'(t)(w) & \end{array} \right)$$

Note that the particular kind of quantification offered in HDL is on time-dependent states. Thus it is more general than usual quantification on rigid variables, as rigid variables can be simulated through states using the predicate

$$\mathrm{constant}(e) \quad \stackrel{\text{def}}{=} \quad \Box(\mathtt{b}.e = \mathtt{e}.e) \ .$$

11.3 Satisfaction, validity, and logical consequence

As in Duration Logic, an HDL formula ϕ *is said to be satisfied by the trajectory* $tr \in \mathit{CTraj}$, denoted $tr \models \phi$, iff any finite prefix of tr satisfies ϕ, i.e. iff any observation $(tr, [0, e])$ with $e \in \mathit{Time}$ satisfies ϕ. Furthermore, a formula ϕ is called *valid* iff any trajectory $tr \in \mathit{CTraj}$ satisfies ϕ. However, due to possible undefinedness of integral expressions, validity is rarely an interesting concept in HDL unless the consequences of undefinedness are carefully controlled using definedness predicates.

An HDL formula ψ is a *logical consequence* of a formula ϕ, denoted $\phi \models \psi$, iff any trajectory satisfying ϕ does also satisfy ψ. As logical consequence is the property to be established in the proofs of the next chapter, we state in the following some laws of logical consequence in HDL. Mostly, we will state them without proofs. The corresponding proofs are often straightforward; some of them can be found in [Rav95b].

As logical consequence is defined as set inclusion between the model classes, it is obviously transitive.

Law 11.1. *If $\phi \models \psi$ and $\psi \models \pi$ then $\phi \models \pi$.*

Within proofs, we will not explicitly mention applications of this law. Instead, we will exploit it by shortly writing chains $\phi_1 \models \phi_2$, $\phi_2 \models \phi_3$, $\phi_3 \models \phi_4$, ... of logical consequences as $\phi_1 \models \phi_2 \models \phi_3 \models \phi_4 \models \ldots$

Another law that we will apply without explicit indication is concerned with local deductions within conjunctions.

Law 11.2. *If $\phi \models \psi$ then $\phi \wedge \pi \models \psi \wedge \pi$.*

The remaining laws are more special to the Duration Calculi, and we will in general mention their application within proofs.

Law 11.3. $(\phi \rightsquigarrow \lceil P \rceil) \;\wedge\; \Box(\psi \Rightarrow \phi) \;\wedge\; \Box(\lceil P \rceil \Rightarrow \lceil Q \rceil) \models \psi \rightsquigarrow \lceil Q \rceil$.

Law 11.4. *If t and t' are constants with $0 < t \leq t'$ then $\lceil P \rceil^t \rightsquigarrow \lceil Q \rceil \models \lceil P \rceil^{t'} \rightsquigarrow \lceil Q \rceil$.*

Furthermore, conventional mathematics can be exploited when reasoning over integral equations in HDL:

Law 11.5. *If $x = f$ is a solution to the differential equation $\dot{x} = e$ then the logical consequence $(\lceil\,\rceil \vee \lceil x = f \rceil) \models \Box\,(\mathtt{e}.x - \mathtt{b}.x = \int e)$ holds in HDL.*

A more sophisticated variant allows to deduce timed leads-to properties of continuous-state systems from conventional mathematical reasoning over differential equations. Therefore, consider predicates in terms of piecewise continuous state variables. If i is such a predicate then the temporal invariant $\lceil i \rceil$ can be defined in HDL as

$$\lceil i \rceil \stackrel{\mathrm{def}}{=} \Box \mathtt{b}.i \ ,$$

where $\mathtt{b}.i$ denotes i with any occurrence of a state variable x being replaced by $\mathtt{b}.x$.

Law 11.6. *Let $\dot{x} = e$ be a differential equation and i and i' predicates in terms of piecewise continuous state variables. Let furthermore be i and i' such that at least one of the two denotes an open subset of Σ. If conventional mathematics shows that after t time units being in states that satisfy i, each solution of $\dot{x} = e$ reaches a state satisfying i', then $\Box\,(\mathtt{e}.x - \mathtt{b}.x = \int e) \models \lceil i \rceil^t \rightsquigarrow \lceil i' \rceil$.*

11.4 Discussion

The extension HDL of Duration Logic with integral equations and inequations on piecewise continuous states provides a simple logic for reasoning about problems that cross the boundary between discrete and continuous systems. Examples are hybrid systems, to which similar formalisms have been frequently applied in the literature, or verification enterprises that attack the semantic gap between the discrete models of digital hardware that are often used in hardware verification and the actual implementation by analog networks. It is the latter, more rarely attacked, problem to which we will apply HDL.

12

A Discrete-State Model of VLSI Dynamics in Embedded Control

Virtually all formalisms for the design of hybrid systems take control automata engaging in instantaneous transitions as the underlying model for discrete components of the system. Implementability of such automata is — except for borderline cases like extreme real-time constraints — usually taken for granted, mainly because switching theory (e.g. [Die71, HP74]) establishes a close link between finite automata and digital circuitry. However, digital hardware is actually implemented by networks of analog components such that the model of instantaneous transitions hopping between discrete states is only an approximation to reality. Thus, we may again ask whether the difference between the idealized model and the actually implemented system dynamics is observable in the field of embedded control and, if so, whether sound models could be built instead.

Unfortunately, the deviation between the idealized instantaneous transitions and the actually implemented transitions is indeed observable in embedded control contexts, as has already been discovered in the late sixties of this century when synchronization failures of independently clocked systems were first encountered. Section 12.1 is devoted to these findings. In the remainder of the chapter we are concerned with a different model for a simple hardware language which is sound with respect to the low-level dynamics of digital hardware, as described by continuous-state circuit-level models of the transistor networks. While some of our analysis of circuit-level models resembles that previously done by R. P. Kurshan and K. L. McMillan to achieve a sound discrete-state (and in their case also discrete-time) model of VLSI that can be subjected to automata-theoretic methods of verification [KM91], we have been able to reduce the size of the resulting state set drastically. This is an advantage when applying automata-theoretic methods of verification, which is the theme of chapter 13.

12.1 Synchronization failure

It was even before the advent of the integrated circuit that technological know-how on how to build and connect digital switching elements had gained sufficient maturity that switching

The material covered by this chapter has previously been available as a ProCoS technical report [Frä95].

theory — an application of finite automata — became a widely accepted formalism for the design of digital systems [Die71, HP74]. The reliance in the formal model was so strong that engineers were both astonished and sceptic when infrequent, but fatal, deviations of implemented devices from switching-theoretic predictions were first reported in the late sixties. The problem — encountered upon connecting independently clocked synchronous systems and having become known as *synchronization failure* [CM73, CW75] — has a simple reason, namely the technical impossibility of instantaneous transitions, and far-reaching consequences, particularly reachability of switching-theoretically unreachable non-transient states.

To see the reason for that unexpected failure of the formal model we may consider a prototypic storage element. In order to be able to store information we need at least two distinct states, which — to prevent information loss due to noise — have to be implemented by sufficiently stable equilibria of the storage device, necessitating a state-potential-dependency of the device akin to that shown in figure 12.1. When performing a state change, energy for driving the bistable element from one stable equilibrium (i.e., a storage state) up to the potential maximum intersecting the two minima has to be supplied *by the stimulus* setting or resetting the bistable element — this is an obvious prerequisite for resistance against noise on the input lines. As the inputs can only supply limited power, dragging a bistable element from one stable state to the other will take time. Hence, a stimulus setting or resetting the bistable element may end before the element has settled in the desired state, possibly leaving the bistable element in an intermediate state from where it does not reach a valid state before its state is next examined, or even in a state not determining in what direction it will relax.

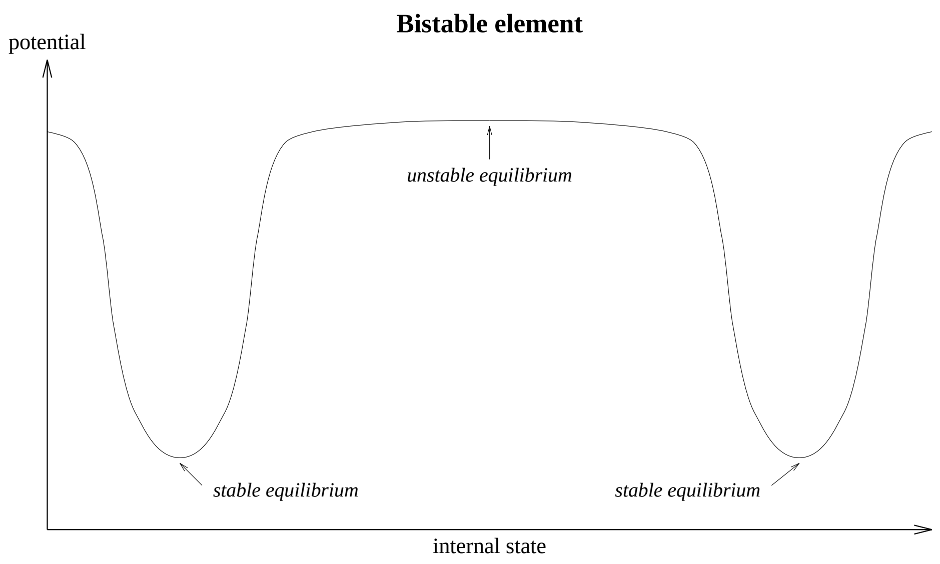


Figure 12.1: Typical state-potential dependency of a bistable element.

Occurrence of such meta-stable states is the key event in synchronization failure: when coupling independently clocked systems, an unsynchronized input may drive a flip-flop to the meta-stable region. Although this may seem to be no more than a non-deterministic assignment to that flip-flop (a situation we have learned to cope with in formal semantics), it is far worse than that. Due to state dependencies, the meta-stable state may be copied to another flip-flop. Thereafter, both flip-flops non-deterministically relax independently of each other, perhaps giving rise to otherwise unreachable states.

As switching theory is a powerful tool for system development, engineers have been worried by that unexpected failure of their formal model and have established design disciplines to circumvent the problem. Basically there are two of them: fully synchronous designs, completely abolishing unsynchronized inputs, and self-timed systems, where progress of the system occurs only after meta-stable states have relaxed. Unfortunately, both disciplines are *not strictly applicable within embedded control*: the synchronous discipline fails because embedded controllers usually have to react upon unsynchronized environment inputs. On the other hand, they have to obey hard real-time constraints, whereas only average performance figures can be asserted upon a self-timed system due to the unbounded relaxation time of meta-stable states.

Beyond these two meta-stability-“free” policies it has been advocated to take the risk of entering meta-stable states as long as their frequency does not exceed the frequency of the anyhow unavoidable thermodynamic error (known as the Mead-criterion [Sei80]). Again, it is quite unclear whether this pragmatic policy is generally applicable to hybrid control, as some components of distributed embedded controllers may have a fairly small state space (reducing the risk of thermodynamic error) yet more unsynchronized inputs than normally encountered in hardware design.

We may conclude that the appropriateness of instantaneous transitions as a model of the dynamic behaviour of discrete components in hybrid systems is doubtful, unless justified through additional theoretical investigation. Yet the instantaneous transitions model is adopted by virtually all proposed formalisms for the design of hybrid systems, as those are usually based on phase-transition systems (e.g. [MP93b, HMP93, Lam93, NOSY93, ACHH93, NK93, NT94, HKPV95]). Hence, the semantic gap between discrete-state models of digital hardware used in such design formalisms and the actual dynamics exhibited by digital hardware has to be bridged.

12.2 A simple hardware language

We take this as a challenge for the notion of implementation correctness that was developed in section 10.4. In the remainder of this chapter, we formalize a behavioural gate-level model of digital circuitry in Duration Logic and verify its soundness against a simple circuit-level model. While this is a prototypic example dealing with simplistic models, it demonstrates that our notion of implementation correctness suffices to bridge the often perceived semantic gap between discrete and continuous models.

In order to demonstrate the shape of the proposed discrete-state behavioural model we will concentrate on a very reduced hardware language being expressive enough to build arbitrary controllers. For the discrete part, ensembles of NAND gates are sufficient, as these are expressively complete. For the interfaces to the environment, 1-Bit sensors sensing a single environment property and 1-Bit actuators establishing a certain property are sufficient: more general sensors and actuators — e.g. n-Bit analog/digital or digital/analog converters — can be built by composing such abstract sensors or actuators, as the abstract 1-Bit devices may observe or establish *arbitrary* state properties over environment states. As we allow arbitrary state properties over environment states to be sensed or established, most of these abstract devices will not be physically realizable. However, this does not affect validity of our subsequent arguments as soundness of the model proposed will particularly hold if the set of realizable sensors and actuators is a subset of those analyzed in the soundness proof: fewer available interfaces weaken the notion of observability even further.

Finally, we need a mechanism to connect the basic devices to form more interesting circuits. Therefore, the ports of these devices are identified by node names. Given a set *Node* of internal node names, the syntax of our simple hardware language is

$$
\begin{array}{lcl}
\langle system \rangle & ::= & \texttt{system}(\langle circuit \rangle) \\
\langle circuit \rangle & ::= & \texttt{nand}_{\langle time \rangle}(\langle node \rangle, \langle node \rangle, \langle node \rangle) \mid \\
 & & \texttt{sens}_{\langle time \rangle}(\langle stateform \rangle, \langle node \rangle) \mid \texttt{act}_{\langle time \rangle}(\langle node \rangle, \langle stateform \rangle) \mid \\
 & & \langle circuit \rangle \,\&\, \langle circuit \rangle \mid \texttt{hide}\ \langle node \rangle\ \texttt{in}\ (\langle circuit \rangle) \\
\langle node \rangle & ::\in & \mathit{Node} \\
\langle time \rangle & ::\in & \mathbb{R}_{>0} \\
\langle stateform \rangle & ::\in & \mathit{EnvironmentFormula}
\end{array}
$$

where *EnvironmentFormula* is the set of state assertions of duration calculus that do only refer to environment variables, i.e. state assertions ϕ with $\mathit{free}(\phi) \cap \mathit{Node} = \emptyset$.

The informal meaning of $\texttt{nand}_t(x, y, z)$ is that of a NAND gate with inputs x and y, controlling output z with a delay of at most t. $\texttt{sens}_t(\phi, x)$ denotes a sensor reflecting validity of ϕ by rising x and falsity of ϕ by making x fall, both with a latency of at most t. Similarly, $\texttt{act}_t(x, \phi)$ is an actuator establishing environment property ϕ if x holds and $\neg\phi$ if $\neg x$ holds, both with a latency of t or less. $c_1 \,\&\, c_2$ denotes composition of circuits c_1 and c_2, i.e. connection of like-named nodes in c_1 and c_2, and $\texttt{hide}\ x\ \texttt{in}\ (c)$ denotes hiding of node x in circuit c. $\texttt{system}(c)$ denotes a complete system, i.e. a circuit where internal nodes are no longer accessible, leaving only the environment interfaces of sensors and actuators visible.[1] In the remainder, the set of terms deducible from the non-terminal $\langle system \rangle$ will be denoted *System* and the set of terms deducible from $\langle circuit \rangle$ will be denoted *Circuit*.

In order to define well-formedness and for hardware-related issues like keeping track of the fan-out in devices we define functions $in : \mathit{Node} \times \mathit{Circuit} \to \mathbb{N}$ and $out : \mathit{Node} \times \mathit{Circuit} \to \mathbb{N}$ which assign the multiplicity with which a node occurs as input and as output

[1]The reader may have expected to find clocks in the hardware language. To avoid unnecessary complexity we have refused from explicitly introducing them. Instead, we consider them to be environmental objects, to be observed and influenced through sensors and actuators. This layout is frequently encountered in formal analyses of the controller design problem as it provides a more homogeneous view of the controller.

in a circuit. Their inductive definition is

$$
\begin{aligned}
in(n, \mathtt{nand}_t(x,y,z)) &\stackrel{\text{def}}{=} \begin{cases} 2 & \text{if } n = x \wedge n = y, \\ 1 & \text{if } (n = x \vee n = y) \wedge x \neq y, \\ 0 & \text{otherwise,} \end{cases} \\
out(n, \mathtt{nand}_t(x,y,z)) &\stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } n = z, \\ 0 & \text{otherwise,} \end{cases} \\
in(n, \mathtt{sens}_t(\phi, x)) &\stackrel{\text{def}}{=} 0 \\
out(n, \mathtt{sens}_t(\phi, x)) &\stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } n = x, \\ 0 & \text{otherwise,} \end{cases} \\
in(n, \mathtt{act}_t(x, \phi)) &\stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } n = x, \\ 0 & \text{otherwise,} \end{cases} \\
out(n, \mathtt{act}_t(x, \phi)) &\stackrel{\text{def}}{=} 0 \ , \\
in(n, c_1 \,\&\, c_2) &\stackrel{\text{def}}{=} in(n, c_1) + in(n, c_2) \ , \\
out(n, c_1 \,\&\, c_2) &\stackrel{\text{def}}{=} out(n, c_1) + out(n, c_2) \ , \\
in(n, \mathtt{hide}\ x\ \mathtt{in}\ (c)) &\stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } n = x, \\ in(n, c) & \text{otherwise,} \end{cases} \\
out(n, \mathtt{hide}\ x\ \mathtt{in}\ (c)) &\stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } n = x, \\ out(n, c) & \text{otherwise.} \end{cases}
\end{aligned}
$$

For convenience, we will use the abbreviations $Out(c) = \{n \in Node \mid out(n,c) > 0\}$ denoting the outputs of a circuit c and $In(c) = \{n \in Node \mid in(n,c) > 0 \wedge n \notin Out(c)\}$ denoting the true, i.e. undriven, inputs of c.

A circuit is well-formed iff no two outputs become connected upon inner circuit compositions and the maximum fanout fo of gates is respected. An exact criterion is that in each subcircuit c' of the circuit every node n occurs at most once as output, i.e. $out(n, c') \leq 1$, and at most fo times as input, i.e. $inp(n, c') \leq fo$. A system $\mathtt{system}(c)$ is *well-formed* iff its enclosed circuit c is well-formed and has neither digital inputs nor digital outputs, i.e. $In(c) = Out(c) = \emptyset$.

12.3 A design-level model in HDL

Although it is sometimes claimed that no two-valued model of hardware can be sound in presence of unsynchronized inputs (P. R. Stephan and R. K. Brayton in [SB93] even claim to have a proof of this fact), we will define an almost classical two-valued design-level model of our gate-level hardware language. Node values are represented by time-dependent Boolean states in our model, and gates establish the appropriate output value with a little delay, provided the inputs are stable.

Technically, each node is interpreted by a like-named discrete HDL state variable, environment variables (occurring in environment formulae) are interpreted by piecewise continuous HDL state variables. The semantic mapping $\mathcal{DL}[\![\cdot]\!] : System \cup Circuit \to Formula$ from circuits and systems to HDL formulae representing their design-level model is defined by the clauses in the remainder of this section:

$$\mathcal{DL}[\![\mathtt{system}(c)]\!] \quad \stackrel{\text{def}}{=} \quad \begin{cases} \mathcal{DL}[\![c]\!], & \text{if } \mathtt{system}(c) \text{ is well-formed,} \\ \mathtt{true} & \text{otherwise,} \end{cases} \tag{12.1}$$

i.e. the behavior of a system is simply determined by its enclosed circuit, given that it is well-formed. For ill-formed systems, the design-level semantics does not allow any prediction.

$$\mathcal{DL}[\![\mathbf{nand}_t(x,y,z)]\!] \quad \stackrel{\text{def}}{=} \quad \Big((\lceil \neg x \rceil^t \vee \lceil \neg y \rceil^t) \leadsto \lceil z \rceil \Big) \;\wedge\; \Big(\lceil x \wedge y \rceil^t \leadsto \lceil \neg z \rceil \Big) \quad , \tag{12.2}$$

i.e. a NAND gate establishes the logically appropriate output value within its assigned delay whenever the inputs are stable during this period. A sensor yields an internal image of a sufficiently persistent external state according to

$$\mathcal{DL}[\![\mathtt{sens}_t(\phi,x)]\!] \quad \stackrel{\text{def}}{=} \quad \Big(\lceil \phi \rceil^t \leadsto \lceil x \rceil \Big) \;\wedge\; \Big(\lceil \neg \phi \rceil^t \leadsto \lceil \neg x \rceil \Big) \quad ,$$

whereas an actuator establishes an externally visible property in response to a sufficiently stable internal state

$$\mathcal{DL}[\![\mathtt{act}_t(x,\phi)]\!] \quad \stackrel{\text{def}}{=} \quad \Big(\lceil x \rceil^t \leadsto \lceil \phi \rceil \Big) \;\wedge\; \Big(\lceil \neg x \rceil^t \leadsto \lceil \neg \phi \rceil \Big) \quad .$$

Circuit composition simply identifies like-named nodes. As the component semantics imposes constraints on node values, well-formed composition is sensibly interpreted as conjunction of those constraints.

$$\mathcal{DL}[\![c_1 \,\&\, c_2]\!] \quad \stackrel{\text{def}}{=} \quad \begin{cases} \mathcal{DL}[\![c_1]\!] \wedge \mathcal{DL}[\![c_2]\!], & \text{if } c_1 \,\&\, c_2 \text{ is well-formed,} \\ \mathtt{true} & \text{otherwise.} \end{cases} \tag{12.3}$$

Again, the model does not allow predictions for ill-formed devices. Note that circuit composition may establish feedback, a form of recursion which due to continuous time cannot be reflected by ω-chain approximation as used in stream-based discrete-time models.

Hiding makes a node value unaccessible to the circuit's context. This is modeled by existential quantification of the corresponding state variable.

$$\mathcal{DL}[\![\mathtt{hide}\ x\ \mathtt{in}\ (c)]\!] \quad \stackrel{\text{def}}{=} \quad \exists x.\mathcal{DL}[\![c]\!] \quad . \tag{12.4}$$

Note that the interpretation of nodes in this design-level model is completely two-valued despite possible meta-stability at the implementation level to be modelled. The rationale behind this design decision is that a two-valued model provides considerable savings in the size of the state space assigned to a circuit compared to a three-valued model. In section 12.5.4 we will show that our model is nevertheless sound. The crucial fact is that within switching transients the model has phases where the node values remain unconstrained. Thus, in contrast to models explicitly modelling the meta-stable range by a distinguished state, we have preferred to reduce it to absence of knowledge.

12.3.1 Relation to switching theory

The model proposed is indeed a conservative extension of switching theory. For combinational circuit structures it is easy to verify the algebraic laws known from switching theory, albeit in a form also supporting reasoning on switch times. E.g., the law

$$\texttt{hide}\ x\ \texttt{in}\ (\texttt{nand}(u,v,x)\,\&\,\texttt{nand}(x,x,w)) = \texttt{and}(u,v,w)$$

finds its counterpart in

$$\texttt{hide}\ x\ \texttt{in}\ (\texttt{nand}_{t_1}(u,v,x)\,\&\,\texttt{nand}_{t_2}(x,x,w)) = \texttt{and}_{t_1+t_2}(u,v,w)\ ,$$

where the semantics of $\texttt{and}_t(u,v,w)$ is defined in the obvious way.

But the HDL model is much more versatile. It outperforms the switching-theoretic model by being able to predict that the non-combinational structure $\texttt{nand}_t(s,q,q')\,\&\,\texttt{nand}_t(r,q',q)$ — an NAND-based RS flip-flop — is state-holding and describes even the influence of preset times on its capability to latch a state, whereas basic switching theory has to treat gates and flip-flops as independent atomic entities subject to restricted composition rules. Furthermore, a straightforward extension of our model including inertia besides upper bounds on switching delay allows to reason algebraically even over sophisticated combinational circuit techniques like wave pipelining — a technique exploiting the switching latency of gates for gaining bandwidth in combinational networks by letting multiple different states coexist in deep combinational networks [LZ93].

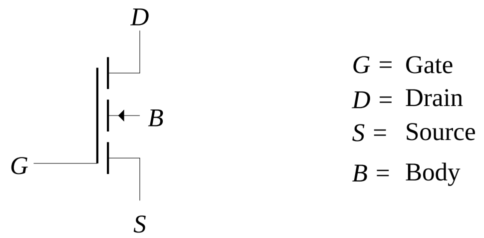
Hence, the HDL semantics of our simple hardware language could be a very interesting model for designing implementations of hybrid control schemes, given that it soundly describes switching circuitry. Therefore we have to show that it is implementable, or — what is worth far more — that it is indeed already implemented by standard digital circuitry. This proof is subject of the next three sections.

12.4 CMOS technology

In the remainder we will provide – and prove correct — a standard CMOS implementation of our simple hardware language. As a starting point we will introduce the basic components of CMOS (complementary metal-oxide semiconductor) technology, namely nMOS and pMOS enhancement mode transistors.

12.4.1 nMOS enhancement mode transistor

The nMOS enhancement mode transistor, which is graphically represented by the IEEE standard symbol



[IEEE78], is the standard component used for switchable pull-down lines in CMOS technology. Whenever its gate-to-source voltage exceeds a threshold voltage U_0, which is positive for enhancement mode devices, it establishes a conduction path between its source and its drain, whereas below threshold voltage source and drain are nearly insulated from each other.[2] The resistance of the conduction path is influenced by both the gate-to-source voltage U_{GS} and the drain-to-source voltage U_{DS}. A simple model giving the approximate dependency of the drain current I_D on these two parameters which suffices for our illustration is the first order model which can be found in most textbooks on semiconductor devices.[3] According to this model the drain-current is approximated as

1. $I_D = 0$ if $U_{GS} \leq U_0$. This operational mode is called *cutoff*.
2. $I_D = -\frac{k}{2}(U_{GS} - U_0)^2$ if $U_0 \leq U_{GS} \leq U_{DS} + U_0$, where k is a geometry-dependent factor (cf. [ML88] for details on k). This operational mode is called *saturated*.
3. $I_D = -k\left(U_{GS} - U_0 - \frac{U_{DS}}{2}\right) U_{DS}$ if $U_{GS} \geq U_{DS} + U_0$, called the *linear* mode.

This model can be expressed in Hybrid Duration Logic by associating with the gate, the source, and the drain of the transistor three real-valued states $\tilde{G}$, $\tilde{S}$, and $\tilde{D}$ recording the voltages of these nodes against an arbitrary reference point (e.g., system ground) and adding a fourth real-valued state $\widetilde{I_D}$ recording the drain current. Then, with $U_{DS} = \tilde{D} - \tilde{S}$ and $U_{GS} = \tilde{G} - \tilde{S}$, above model can be reformulated as

$$\mathtt{nmos}(\tilde{G}, \tilde{S}, \tilde{D}) \stackrel{\text{def}}{=}$$
$$\Box \left(\begin{array}{rcl} \lceil \tilde{G} - \tilde{S} \leq U_0 \rceil & \Rightarrow & \lceil \widetilde{I_D} = 0 \rceil \wedge \\ \lceil \tilde{G} - \tilde{S} \geq U_0 \wedge \tilde{G} \leq \tilde{D} + U_0 \rceil & \Rightarrow & \lceil \widetilde{I_D} = -\frac{k}{2}(\tilde{G} - \tilde{S} - U_0)^2 \rceil \wedge \\ \lceil \tilde{G} \geq \tilde{D} + U_0 \rceil & \Rightarrow & \lceil \widetilde{I_D} = -k\left(\tilde{G} - U_0 - \frac{\tilde{D}+\tilde{S}}{2}\right)(\tilde{D} - \tilde{S}) \rceil \end{array} \right)$$

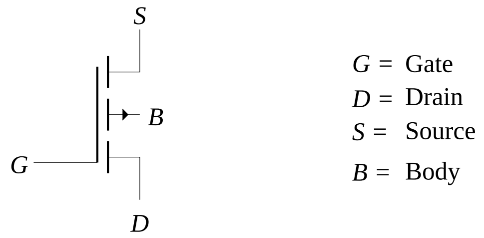
Beyond being a voltage-controlled resistor, a MOS transistor is characterized by a set of voltage-dependent capacitances between its nodes. To keep things simple we abstract from the exact details by assuming that we can assert a *maximum* load capacitance C_L it imposes on the device driving its gate.

12.4.2 pMOS enhancement mode transistor

The pMOS enhancement mode transistor, graphically represented by the IEEE standard symbol

[2] Because of the positive threshold voltage of enhancement mode transistors, insulation between source and drain particularly occurs if the gate and the source potentials agree. For this reason, enhancement mode transistors are sometimes also called *normally-off type* transistors.

[3] The interested reader may consult e.g. [ML88, p. 291 ff] or [Sin94, p. 424 ff] for a derivation of this model from device physics and geometry. [Sno89] gives an overview over various other kinds of field-effect device models.



is the standard component used for switchable pull-up lines in CMOS technology. Whenever its source-to-gate voltage[4] exceeds the threshold voltage U_0 it establishes a conduction path between its source and its drain, whereas below threshold voltage source and drain are nearly insulated from each other. As furthermore, in contrast to the standard operational mode of nMOS transistors, its source potential should normally be higher than its drain potential, it behaves similar to its nMOS counterpart, except for a consistent change in polarity, giving the first order model

1. $I_D = 0$ if $U_{SG} \leq U_0$. (*Cutoff*)
2. $I_D = \frac{k}{2}(U_{SG} - U_0)^2$ if $U_0 \leq U_{SG} \leq U_{SD} + U_0$. (*Saturation*)
3. $I_D = k\left(U_{SG} - U_0 - \frac{U_{SD}}{2}\right) U_{SD}$ if $U_{SG} \geq U_{SD} + U_0$. (*Linear operation*)

and the corresponding Duration Logic description

$$\mathtt{pmos}(\tilde{G}, \tilde{S}, \tilde{D}) \stackrel{\text{def}}{=}$$
$$\Box \left(\begin{array}{rcl} \lceil \tilde{S} - \tilde{G} \leq U_0 \rceil & \Rightarrow & \lceil \widetilde{I_D} = 0 \rceil \wedge \\ \lceil \tilde{S} - \tilde{G} \geq U_0 \wedge \tilde{G} \geq \tilde{D} - U_0 \rceil & \Rightarrow & \lceil \widetilde{I_D} = \frac{k}{2}(\tilde{S} - \tilde{G} - U_0)^2 \rceil \wedge \\ \lceil \tilde{G} \leq \tilde{D} - U_0 \rceil & \Rightarrow & \lceil \widetilde{I_D} = k\left(-\tilde{G} - U_0 + \frac{\tilde{D}+\tilde{S}}{2}\right)(\tilde{S} - \tilde{D}) \rceil \end{array} \right) .$$

12.4.3 CMOS NAND gate

A CMOS NAND gate can be built by putting two pMOS transistors — each having one of the inputs connected to its gate — in parallel as a pull-up network and two nMOS transistors — again each having one of the inputs connected to its gate — in series as a pull-down network, as shown in figure 12.2 on the following page [WS63, GD85]. The net effect is that the pull-up network is activated whenever at least one of the inputs is low, resulting in high output voltage, and that the pull-down network is activated when both inputs are high, resulting in low output voltage.

A more thorough investigation of the network behavior can be deduced from the models of nMOS and pMOS enhancement transistors by appropriately instantiating them and formalizing the connection structure by adding Kirchhoff's laws. Hiding the internal node

[4]Note the reversed polarity when compared to the nMOS transistor.

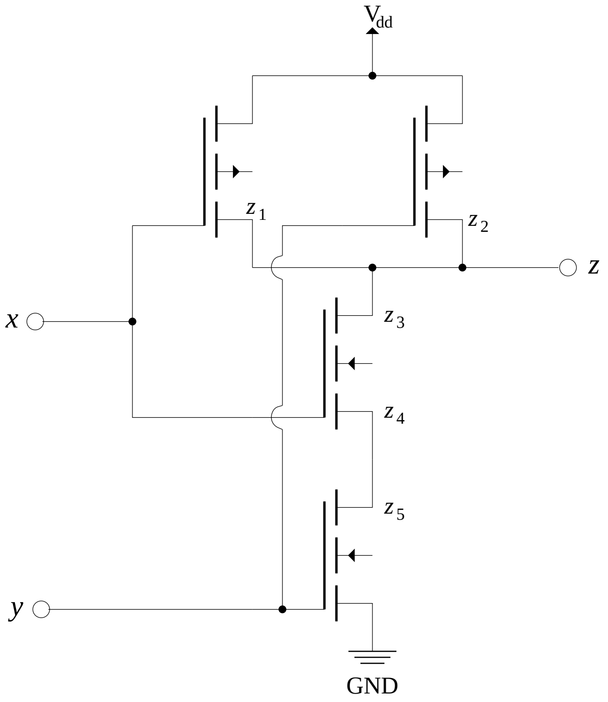


Figure 12.2: CMOS NAND gate.

voltages $\tilde{z}_1$ to $\tilde{z}_5$ and the drain currents $\widetilde{I_{z_1}}$ to $\widetilde{I_{z_5}}$ by existential quantification, this gives the dependency

$$\begin{aligned}
&\exists \tilde{z}_1, \tilde{z}_2, \tilde{z}_3, \tilde{z}_4, \tilde{z}_5, \widetilde{I_{z_1}}, \widetilde{I_{z_2}}, \widetilde{I_{z_3}}, \widetilde{I_{z_5}}. \\
&\quad \mathtt{pmos}(\tilde{x}, \mathrm{V_{dd}}, \tilde{z}_1) \;\wedge\; \mathtt{pmos}(\tilde{y}, \mathrm{V_{dd}}, \tilde{z}_2) \;\wedge \\
&\quad \mathtt{nmos}(\tilde{x}, \tilde{z}_4, \tilde{z}_3) \;\wedge\; \mathtt{nmos}(\tilde{y}, \mathrm{GND}, \tilde{z}_5) \;\wedge \\
&\quad \Box(\lceil\,\rceil \vee \lceil \tilde{z} = \tilde{z}_1 = \tilde{z}_2 = \tilde{z}_3 \;\wedge\; \widetilde{I_z} = \widetilde{I_{z_1}} + \widetilde{I_{z_2}} + \widetilde{I_{z_3}} \rceil) \;\wedge \\
&\quad \Box(\lceil\,\rceil \vee \lceil \tilde{z}_4 = \tilde{z}_5 \;\wedge\; \widetilde{I_{z_3}} = \widetilde{I_{z_5}} \rceil)
\end{aligned} \tag{12.5}$$

of the output current $\widetilde{I_z}$ on input node voltages $\tilde{x}$ and $\tilde{y}$.

In order to determine the dynamic behavior of the CMOS NAND gate we examine the effect of driving a (possibly non-constant) load capacitance $\widetilde{C_z}$ by node $\tilde{z}$. The obvious result is that $\tilde{z}$'s voltage will change with a rate proportional to $\widetilde{I_z}$, where the proportionality factor is $\widetilde{C_z}^{-1}$. In Hybrid Duration Logic this is expressed as

$$\Box\left(\mathsf{e}.\tilde{z} - \mathsf{b}.\tilde{z} = \int \frac{\widetilde{I_z}}{\widetilde{C_z}}\right) \quad . \tag{12.6}$$

In the circuits we are going to produce, each node drives at most *fo* gate inputs, where *fo* is a small natural number, the maximal fan-out. As each input of a CMOS gate consists of at most two transistor gates (one nMOS transistor gate and one pMOS transistor gate) each node drives at most $2fo$ transistor gate capacitances, each of which is bounded by the maximum load capacitance C_L. Thus, for the actual load capacitance $\widetilde{C_z}$ driven by node $\tilde{z}$,

$$\Box\left(\lceil\rceil \vee \lceil \widetilde{C_z} \leq 2\,fo\,C_L \rceil\right) \tag{12.7}$$

holds.

We can combine properties (12.5), (12.6), and (12.7) to form the circuit-level model $\mathcal{CL}[\![\text{NAND}(\cdot,\cdot,\cdot)]\!]$ of CMOS NAND gate behavior in the circuits considered, obtaining

$$\mathcal{CL}[\![\text{NAND}(\tilde{x},\tilde{y},\tilde{z})]\!] \quad \stackrel{\text{def}}{=} \quad (12.5) \ \wedge \ (12.6) \ \wedge \ (12.7)$$

as a model of the dynamics of a CMOS NAND gate with inputs $\tilde{x}$ and $\tilde{y}$ and output $\tilde{z}$. This model (or indeed the derived property (12.8) below) will form the basis of the proofs concerning implementation of the abstract semantics by CMOS structures in the remainder of this chapter.

Using a similar analysis to that given for the CMOS inverter in [ML88, p. 386 f] one obtains a worst case estimation for the rise time of the CMOS NAND gate (i.e. the time taken for $\tilde{z}$ to travel from an arbitrary voltage in the range $[\text{GND}, \text{GND} + U_0]$, called the *low range*, to the range $[\text{V}_{\text{dd}} - U_0, \text{V}_{\text{dd}}]$, called the *high range*, given that at least one of the inputs $\tilde{x}$ and $\tilde{y}$ is permanently in the low range) of

$$\begin{aligned} t_{\text{rise}} \quad &\leq \quad \frac{\widetilde{C_z}}{2k} \frac{\ln\left(2\frac{\text{V}_{\text{dd}}}{U_0} - 3\right)}{\text{V}_{\text{dd}} - 2U_0} \\ &\leq \quad \frac{fo\,C_L}{k} \frac{\ln\left(2\frac{\text{V}_{\text{dd}}}{U_0} - 3\right)}{\text{V}_{\text{dd}} - 2U_0} \quad . \end{aligned}$$

A similar analysis for the worst case fall time (i.e. the maximum time taken for $\tilde{z}$ to travel from the high range to the low range, given that both inputs are permanently in the high range) gives

$$t_{\text{fall}} \quad \leq \quad \frac{2fo\,C_L}{k} \frac{\ln\left(2\frac{\text{V}_{\text{dd}}}{U_0} - 3\right)}{\text{V}_{\text{dd}} - 2U_0} \quad .$$

Hence, using law 11.6 of HDL we can validly deduce

$$\begin{aligned} \mathcal{CL}[\![\text{NAND}(\tilde{x},\tilde{y},\tilde{z})]\!] \quad \models \quad & \left(\left(\lceil \text{lo}(\tilde{x}) \rceil^{t_{\text{nand}}} \vee \lceil \text{lo}(\tilde{y}) \rceil^{t_{\text{nand}}}\right) \leadsto \lceil \text{hi}(\tilde{z}) \rceil\right) \ \wedge \\ & \left(\lceil \text{hi}(\tilde{x}) \wedge \text{hi}(\tilde{y}) \rceil^{t_{\text{nand}}} \leadsto \lceil \text{lo}(\tilde{z}) \rceil\right) \quad , \end{aligned} \tag{12.8}$$

where $\text{lo}(\tilde{v})$ abbreviates $\tilde{v} < \text{GND} + U_0$, $\text{hi}(\tilde{v})$ abbreviates $\tilde{v} > \text{V}_{\text{dd}} - U_0$, and t_{nand} is an arbitrary constant safely bounding the rise and fall times, i.e. satisfying

$$t_{\text{nand}} \quad \geq \quad \frac{2fo\,C_L}{k} \frac{\ln\left(2\frac{\text{V}_{\text{dd}}}{U_0} - 3\right)}{\text{V}_{\text{dd}} - 2U_0} \quad .$$

Indeed, property (12.8) is the only property of the implemented NAND gate that is used in the remainder, such that our claim that the further verification is largely independent from the simplifications of the particular transistor model should seem trustworthy.

Although the right hand side of (12.8) may seem to correspond closely to the abstract NAND semantics (12.2), a subtle and important difference is to be noted: the abstract node values in (12.2) are Boolean, whereas (12.8) gives in effect a three-valued description — a node state $\tilde{x}$ may in a time instant satisfy the state assertion $\mathrm{lo}(\tilde{x})$ or the state assertion $\mathrm{hi}(\tilde{x})$ or none of both. Hence, the meta-stable range is still explicitly present in (12.8) whereas it can only be represented by instability of a properly Boolean state in (12.2).

12.4.4 Sensors and actuators

We will not go into the details of sensor and actuator design here. But following the lines of the argument put forward for property (12.8) of the CMOS NAND gate, it seems reasonable that for any implemented sensor $\textsc{sens}(\phi, \tilde{x})$, sensing environment property ϕ and reflecting it by switching internal node $\tilde{x}$, the constructing engineer should be able to assert dynamic properties $\mathcal{CL}[\![\textsc{sens}(\phi, \tilde{x})]\!]$ of the device which satisfy

$$\mathcal{CL}[\![\textsc{sens}(\phi, \tilde{x})]\!] \quad \models \quad \left(\lceil \phi \rceil^{t_{\mathrm{sens}}} \leadsto \lceil \mathrm{hi}(\tilde{x}) \rceil\right) \ \wedge \ \left(\lceil \neg\phi \rceil^{t_{\mathrm{sens}}} \leadsto \lceil \mathrm{lo}(\tilde{x}) \rceil\right) \tag{12.9}$$

for a sufficiently large time constant t_{sens}. For realistic sensors this time constant will be much larger than t_{nand}, but this is no prerequisite for correctness of the CMOS implementation of the abstract hardware language encountered in section 12.5.3.

Similarly, for an implemented actuator $\textsc{act}(\tilde{x}, \phi)$, establishing environment property ϕ depending on node $\tilde{x}$, the engineer should be able to assert dynamic properties $\mathcal{CL}[\![\textsc{act}(\tilde{x}, \phi)]\!]$ satisfying

$$\mathcal{CL}[\![\textsc{act}(\tilde{x}, \phi)]\!] \quad \models \quad \left(\lceil \mathrm{hi}(\tilde{x}) \rceil^{t_{\mathrm{act}}} \leadsto \lceil \phi \rceil\right) \ \wedge \ \left(\lceil \mathrm{lo}(\tilde{x}) \rceil^{t_{\mathrm{act}}} \leadsto \lceil \neg\phi \rceil\right) \tag{12.10}$$

for a sufficiently large time constant t_{act}, presumably even larger than t_{sens}.

12.4.5 Connecting CMOS devices to form larger circuits

Similarly to what was done when constructing the CMOS NAND gate from field-effect transistors, we can connect CMOS devices through shared nodes to form more interesting circuits. The behavior of the resultant device can be determined by applying Kirchhoff's laws and taking care of the additivity of node capacitances upon connection.[5] If only like-named nodes are connected, Kirchhoff's voltage law is automatically enforced by taking the conjunction of the component models, as the variables representing the node voltages become simply identified. Furthermore, if we obey the discipline that a node can be an output of at most one of the components to be connected, i.e. that there is at most one

[5]Clearly, the capacitance of the connection line has also to be taken into account. For simplicity, we assume that it has already been reflected in the worst-case node capacitance C_L, which is a bit overly optimistic for realistic structures, as connection lines do not only add to the capacitance, but form complicated RC networks.

current injector per node, then Kirchhoff's current law is also taken care of by conjunction of the component models, as the node current is only constrained in the appropriate component model.

A more subtle problem upon modelling circuit composition is the additivity of node capacitances. But as the component models encountered so far did not state more about node capacitances than the global invariant (12.7) that remains true within all the compositions we are going to produce, conjunction of the component models will not allow invalid deductions concerning node capacitances.

Hence, modelling circuit composition by model conjunction, as in $\mathcal{CL}[\![c_1 \,\&\, c_2]\!] = \mathcal{CL}[\![c_1]\!] \wedge \mathcal{CL}[\![c_2]\!]$, is in accordance with the laws of electrical networks if we obey the connection discipline that

(a) each node occurs at most once as an output in $c_1 \,\&\, c_2$, yielding at most one current source per node, and

(b) each node occurs at most *fo* times as an input in $c_1 \,\&\, c_2$, limiting the fan-out of each gate output to *fo* and thus making invariant (12.7) true.

Thus, we may safely define

$$\mathcal{CL}[\![c_1 \,\&\, c_2]\!] \stackrel{\text{def}}{=} \begin{cases} \mathcal{CL}[\![c_1]\!] \wedge \mathcal{CL}[\![c_2]\!] & \text{if (a)} \wedge \text{(b)}, \\ \texttt{true} & \text{otherwise}, \end{cases} \tag{12.11}$$

thereby explicitly reflecting the dependency of the model on the connection discipline.

12.4.6 Hiding internal nodes of circuits

Sometimes it makes sense to explicitly state that a certain node is no longer accessible to circuit composition. It should be obvious that the effect of making a node unaccessible is adequately reflected by existentially quantifying all its characteristics, i.e. its potential, its current, and its load capacitance:

$$\mathcal{CL}[\![\text{HIDE } \tilde{x} \text{ IN } (c)]\!] \stackrel{\text{def}}{=} \exists \tilde{x}, \widetilde{I_x}, \widetilde{C_x}.\mathcal{CL}[\![c]\!] \ . \tag{12.12}$$

12.4.7 Complete system

A system in CMOS technology is just a CMOS circuit and hence its behavior coincides with that of the circuit enclosed. However, we will only postulate that well-formed systems. The behaviour of ill-formed systems is deliberately left open which certainly is a physically adequate description.

$$\mathcal{CL}[\![\text{SYSTEM}(c)]\!] \stackrel{\text{def}}{=} \begin{cases} \mathcal{CL}[\![c]\!], & \text{if SYSTEM}(c) \text{ is well-formed}, \\ \texttt{true} & \text{otherwise}. \end{cases} \tag{12.13}$$

We will only consider such CMOS circuits as being *well-formed* or *complete systems* which are packaged such that internal nodes are no longer accessible, leaving only the sensor and actuator interfaces as connections to the environment. Hence, a well-formed system is a circuit where hiding has been applied until no internal nodes remained accessible, i.e. a circuit c with $In(c) = Out(c) = \emptyset$, where the sets $In(c)$ and $Out(c)$ of input and output nodes of CMOS circuits are similarly defined as for the abstract hardware language.

12.5 Soundness of the design model with respect to circuit-level dynamics

The essence of any development activity is to make sure that delivered devices meet the expectations. Formal models are intended to help meet this objective. Hence, we may call a formal model of an implementation technology *sound* if whenever it predicts that a device meets the expectations then this is true of the implemented device. Our claim is that the discrete-state model of our hardware language is sound, i.e. faithfully models the CMOS implementation. The current section is devoted to giving this claim support through formal verification, using the techniques of section 10.4.

The notion of refinement proposed in chapter 10 crucially depends on the notion of expectations that might be put on a complete design. Therefore, an application of this theory starts with

(a) defining what expectations, i.e. success criteria for the design, might be placed on systems, and

(b) defining what satisfaction of an expectation means.

12.5.1 Success criteria

Due to the rather close correspondence of our abstract semantics and the implementation technology we need not constrain the choice of success criteria (or expectations) that may be placed on complete systems: we allow to use arbitrary formulae of Hybrid Duration Logic, i.e.

$$S \stackrel{\text{def}}{=} \{\Phi \mid \Phi \text{ is a formula of HDL}\} \ .$$

Note that these are indeed *all* statements that can be reasonably made upon an embedded controller in Hybrid Duration Logic, such that refinement with respect to this set of success criteria means that *any HDL property valid for a complete design is retained by its implementation.*

12.5.2 Satisfaction of expectations by complete systems

In chapter 10, the theory of refinement was developed in the context of denotational semantics, where denotational order qualifies as a notion of satisfaction. While we use predicative instead of denotational semantics in the present chapter, the definition of the satisfaction relation $\mathtt{sat}$ in our context is just as obvious: a system π of the hardware language satisfies a success criterion $s \in S$ iff validity of s is a logical consequence of π's design-level semantics, i.e.

$$\pi \ \mathtt{sat}\ s \quad \text{iff} \quad \mathcal{DL}[\![\pi]\!] \models s \ .$$

Similarly, a CMOS system ψ satisfies a success criterion $s \in S$ iff validity of s is a logical consequence of ψ's dynamic behaviour, i.e.

$$\psi \ \mathtt{sat}\ s \quad \text{iff} \quad \mathcal{CL}[\![\psi]\!] \models s \ .$$

The connection to the denotationally based refinement theory of chapter 10 is that formulae in predicative semantics denotes sets, namely the sets of models satisfying the formulae, and that the satisfaction relation $\models$ corresponds to set inclusion between the model classes such that the denotational order $\sqsubseteq$ can be taken to be reverse set inclusion $\supseteq$. Hence, the theory of chapter 10 is directly applicable to predicative semantics, provided that the side condition that all operators be monotonic with respect to denotational order (i.e. logical consequence) is satisfied. This monotonicity is straightforward to check in our setting.

12.5.3 CMOS implementation of the simple hardware language

Before we can sensibly say that the design-level semantics is sound with respect to the implementation, we have to formalize what we consider to be its implementation. Formally, this is a compilation of terms of the design-level hardware language to CMOS structures, albeit a straightforward one. It simply maps the abstract devices to their concrete counterparts. I.e., the compiler C is the relation induced by $(C_{\text{type}}, C_{\text{base}}, C_{\text{struct}})$, where C_{type} defines that abstract nodes are mapped to concrete nodes (i.e. $C_{\text{type}}(\mathit{Node}) = \widetilde{\mathit{Node}}$), abstract circuits to concrete circuits ($C_{\text{type}}(\mathit{Circuit}) = \widetilde{\mathit{Circuit}}$), and abstract systems to concrete systems ($C_{\text{type}}(\mathit{System}) = \widetilde{\mathit{System}}$). The base relation C_{base} is only concerned with nodes, and the only fact that we require of it is that it is an injective function, possibly partial:

$$C_{\text{base}} : \mathit{Node} \xrightarrow{\text{partial}} \widetilde{\mathit{Node}}, \text{ injective.} \tag{12.14}$$

The structural rules are simply concerned with mapping abstract constructs to their concrete realizations, thereby checking realizability of the desired switching latency. Thus, the structural relation C_{struct} is indeed a partial function, and we will use the more convenient functional notation for defining it.

$$C_{\text{struct}}(\mathtt{nand}_t(\sqcup_1, \sqcup_2, \sqcup_3)) \stackrel{\text{def}}{=} \begin{cases} \text{NAND}(\sqcup_1, \sqcup_2, \sqcup_3) & \text{if } t \geq t_{\text{nand}}, \\ \text{undefined} & \text{otherwise,} \end{cases} \tag{12.15}$$

$$C_{\text{struct}}(\mathtt{sens}_t(\phi, \sqcup)) \stackrel{\text{def}}{=} \begin{cases} \text{SENS}(\phi, \sqcup) & \text{if } t \geq t_{\text{sens}}, \\ \text{undefined} & \text{otherwise,} \end{cases} \tag{12.16}$$

$$C_{\text{struct}}(\mathtt{act}_t(\sqcup, \phi)) \stackrel{\text{def}}{=} \begin{cases} \text{ACT}(\sqcup, \phi) & \text{if } t \geq t_{\text{act}}, \\ \text{undefined} & \text{otherwise,} \end{cases} \tag{12.17}$$

$$C_{\text{struct}}(\sqcup_1 \,\&\, \sqcup_2) \stackrel{\text{def}}{=} \sqcup_1 \,\&\, \sqcup_2 \ , \tag{12.18}$$

$$C_{\text{struct}}(\texttt{hide } \sqcup_1 \texttt{ in } (\sqcup_2)) \stackrel{\text{def}}{=} \text{HIDE } \sqcup_1 \text{ IN } (\sqcup_2) \ , \tag{12.19}$$

$$C_{\text{struct}}(\texttt{system}(\sqcup)) \stackrel{\text{def}}{=} \text{SYSTEM}(\sqcup) \ . \tag{12.20}$$

We claim that this compilation scheme is correct with respect to the behavioural models of source and target level:

Lemma 12.1. *The relation C induced by $(C_{\text{type}}, C_{\text{base}}, C_{\text{struct}})$ is a correct compiler for the simple hardware language to CMOS circuitry, as formalized by the behavioural model $\mathcal{CL}[\![\cdot]\!]$.*

The proof is subject of the next section.

12.5.4 Verification of compilation to CMOS circuitry

According to the verification rule for compilers put forward in theorem 10.21, verification of C amounts to identifying a relation I between terms of the hardware language and CMOS structures satisfying conditions (i) to (iv) of lemma 10.18, i.e.

i) if $(P, Q) \in I$ and P has type t then Q has type $C_{\text{type}}(t)$,

ii) for each abstract system π and each concrete system ψ with $(\pi, \psi) \in I$ it holds that π **sat** s implies ψ **sat** s for each $s \in S$,

iii) $C_{\text{base}} \subseteq I$, and

iv) for each operator $o : t_1, \ldots, t_n \to t$ in O and all $\mathcal{L}$-terms $P_1, \ldots, P_n$ of types $t_1, \ldots, t_n$ and all $\tilde{\mathcal{L}}$-terms $Q_1, \ldots, Q_n$ of types $C_{\text{type}}(t_1), \ldots, C_{\text{type}}(t_n)$ the implication

$$(\forall i \in \{1, \ldots, n\} \,.\, (P_i, Q_i) \in I\ ) \Rightarrow (o(P_1, \ldots, P_n), e(Q_1, \ldots, Q_n)) \in I$$

holds for each $(o, e) \in C_{\text{struct}}$.

In the course of the verification we will define I by successively identifying three relations I_{Node} (relating abstract and concrete nodes), $I_{Circuit}$ (relating abstract and concrete circuits), and I_{System} (relating abstract and concrete systems). I is the union of these three relations. Due to this style of definition, verification condition (i) is trivially discharged.

Node compilation

For nodes we simply define the inductive relation I_{Node} to be equivalent to the compilation function: $I_{Node} \stackrel{\text{def}}{=} C_{\text{base}}$.

This may be worrying to the reader as it trivially discharges proof obligation (iii), making the base cases of compilation *by definition* correct. But by soundness of the inductive proof rule this is not at all harmful. Instead, the proof obligation is just moved to more appropriate places. As the proof — in a whole — establishes that I is a subrelation of the refinement relation, the completed proof will show that also I_{Node} and thus node compilation is a subrelation of refinement, as desired. No local proof is required due to the soundness of the inductive proof rule.

Circuit compilation

The correctness relation $I_{Circuit}$ for circuits is a bit more tricky to define. We conjunctively combine the model of the concrete circuit with an abstraction relation that collapses to `true` for complete systems, and show that this conjunction entails the semantics of the abstract device. Therefore, we define

$$c\, I_{Circuit}\, \tilde{c} \qquad \text{iff} \qquad \mathcal{CL}[\![\tilde{c}]\!] \;\wedge\; abs_{Nodes(c)} \models \mathcal{DL}[\![c]\!] \;,$$

where $Nodes(c) \stackrel{\text{def}}{=} In(c) \cup Out(c)$. The abstraction relation coupling abstract and concrete node values is, for each $M \subseteq Node$ and each $n \in Node$ and $\tilde{n} \in \widetilde{Node}$ with $n\, I_{Node}\, \tilde{n}$, defined by

$$\begin{aligned}
abs_M &\stackrel{\text{def}}{=} \bigwedge_{n \in M} abs_n \;, \\
abs_n &\stackrel{\text{def}}{=} \Box \left(\begin{array}{rcll} \lceil \mathrm{lo}(\tilde{n}) \rceil & \Rightarrow & \lceil \neg n \rceil & \wedge \\ \lceil \mathrm{hi}(\tilde{n}) \rceil & \Rightarrow & \lceil n \rceil & \wedge \\ (\neg(\lceil \mathrm{lo}(\tilde{n}) \rceil \vee \lceil \mathrm{hi}(\tilde{n}) \rceil) \;\wedge\; (\lceil \neg n \rceil \vee \lceil n \rceil)) & \Rightarrow & l < t^* & \end{array} \right) , \\
t^* &\stackrel{\text{def}}{=} \min(t_{\mathrm{nand}}, t_{\mathrm{sens}}, t_{\mathrm{act}}) \;.
\end{aligned}$$

The last conjunct of abs_n expresses that whenever the concrete node $\tilde{n}$ does not stably encode a truth value then there are behaviors of the abstract node n that change state at least once within every t^* time units. As t^* is smaller than the switching delays of the concrete devices this means that the abstract level also does not encode a truth value that can be recognized by any gate.

Having defined the correctness relation for circuits, we are now in the position to discharge proof obligation (iv) for the five structural rules (12.15) to (12.19).

Correctness of implementing the abstract NAND by a CMOS NAND gate. Let $x, y, z \in Node$ and $x\, I_{Node}\, \tilde{x}$, $y\, I_{Node}\, \tilde{y}$, $z\, I_{Node}\, \tilde{z}$ and $t \geq t_{\mathrm{nand}}$. First note that

$$abs_n \;\models\; \Box \left(\begin{array}{rcl} (\lceil \neg n \rceil \;\wedge\; l \geq t^*) & \Rightarrow & \lceil \mathrm{lo}(\tilde{n}) \rceil \wedge \\ (\lceil n \rceil \;\wedge\; l \geq t^*) & \Rightarrow & \lceil \mathrm{hi}(\tilde{n}) \rceil \end{array} \right) \tag{12.21}$$

for any node $n \in Node$. Thus we can deduce

$$\begin{aligned}
& \mathcal{CL}[\![\mathrm{NAND}(\tilde{x}, \tilde{y}, \tilde{z})]\!] \;\wedge\; abs_{Nodes(\mathtt{nand}_t(x,y,z))} \\
\models\; & \{Nodes(\mathtt{nand}_t(x,y,z)) = \{x, y, z\}.\} \\
& \mathcal{CL}[\![\mathrm{NAND}(\tilde{x}, \tilde{y}, \tilde{z})]\!] \wedge abs_x \wedge abs_y \wedge abs_z \\
\models\; & \{(12.8).\} \\
& \big(\big(\lceil \mathrm{lo}(\tilde{x}) \rceil^{t_{\mathrm{nand}}} \vee \lceil \mathrm{lo}(\tilde{y}) \rceil^{t_{\mathrm{nand}}} \big) \leadsto \lceil \mathrm{hi}(\tilde{z}) \rceil \big) \;\wedge\; \big(\lceil \mathrm{hi}(\tilde{x}) \wedge \mathrm{hi}(\tilde{y}) \rceil^{t_{\mathrm{nand}}} \leadsto \lceil \mathrm{lo}(\tilde{z}) \rceil \big) \wedge \\
& abs_x \wedge abs_y \wedge abs_z
\end{aligned}$$

$$\begin{aligned}
\models\ & \{(12.21), \text{ applied to nodes } x \text{ and } y.\} \\
& \big((\lceil \mathrm{lo}(\tilde{x})\rceil^{t_{\mathrm{nand}}} \vee \lceil \mathrm{lo}(\tilde{y})\rceil^{t_{\mathrm{nand}}}) \leadsto \lceil \mathrm{hi}(\tilde{z})\rceil\big) \;\wedge\; \big(\lceil \mathrm{hi}(\tilde{x}) \wedge \mathrm{hi}(\tilde{y})\rceil^{t_{\mathrm{nand}}} \leadsto \lceil \mathrm{lo}(\tilde{z})\rceil\big) \;\wedge \\
& \Box\begin{pmatrix} (\lceil \neg x\rceil \;\wedge\; l \geq t^*) & \Rightarrow & \lceil \mathrm{lo}(\tilde{x})\rceil \;\wedge \\ (\lceil x\rceil \;\wedge\; l \geq t^*) & \Rightarrow & \lceil \mathrm{hi}(\tilde{x})\rceil \end{pmatrix} \wedge \Box\begin{pmatrix} (\lceil \neg y\rceil \;\wedge\; l \geq t^*) & \Rightarrow & \lceil \mathrm{lo}(\tilde{y})\rceil \;\wedge \\ (\lceil y\rceil \;\wedge\; l \geq t^*) & \Rightarrow & \lceil \mathrm{hi}(\tilde{y})\rceil \end{pmatrix} \wedge \\
& \mathit{abs}_z \\[1ex]
\models\ & \{t^* \leq t_{\mathrm{nand}},\ \phi \Rightarrow \psi \models \phi \wedge \xi \Rightarrow \psi \wedge \xi.\} \\
& \big((\lceil \mathrm{lo}(\tilde{x})\rceil^{t_{\mathrm{nand}}} \vee \lceil \mathrm{lo}(\tilde{y})\rceil^{t_{\mathrm{nand}}}) \leadsto \lceil \mathrm{hi}(\tilde{z})\rceil\big) \;\wedge\; \big(\lceil \mathrm{hi}(\tilde{x}) \wedge \mathrm{hi}(\tilde{y})\rceil^{t_{\mathrm{nand}}} \leadsto \lceil \mathrm{lo}(\tilde{z})\rceil\big) \;\wedge \\
& \Box\begin{pmatrix} \lceil \neg x\rceil^{t_{\mathrm{nand}}} & \Rightarrow & \lceil \mathrm{lo}(\tilde{x})\rceil \;\wedge \\ \lceil x\rceil^{t_{\mathrm{nand}}} & \Rightarrow & \lceil \mathrm{hi}(\tilde{x})\rceil \end{pmatrix} \wedge \Box\begin{pmatrix} \lceil \neg y\rceil^{t_{\mathrm{nand}}} & \Rightarrow & \lceil \mathrm{lo}(\tilde{y})\rceil \;\wedge \\ \lceil y\rceil^{t_{\mathrm{nand}}} & \Rightarrow & \lceil \mathrm{hi}(\tilde{y})\rceil \end{pmatrix} \wedge \\
& \mathit{abs}_z \\[1ex]
\models\ & \{\text{Law } 11.3.\} \\
& \big((\lceil \neg x\rceil^{t_{\mathrm{nand}}} \vee \lceil \neg y\rceil^{t_{\mathrm{nand}}}) \leadsto \lceil \mathrm{hi}(\tilde{z})\rceil\big) \;\wedge\; \big(\lceil x \wedge y\rceil^{t_{\mathrm{nand}}} \leadsto \lceil \mathrm{lo}(\tilde{z})\rceil\big) \;\wedge\; \mathit{abs}_z \\[1ex]
\models\ & \{\text{Definition of } \mathit{abs}_z.\} \\
& \big((\lceil \neg x\rceil^{t_{\mathrm{nand}}} \vee \lceil \neg y\rceil^{t_{\mathrm{nand}}}) \leadsto \lceil \mathrm{hi}(\tilde{z})\rceil\big) \;\wedge\; \big(\lceil x \wedge y\rceil^{t_{\mathrm{nand}}} \leadsto \lceil \mathrm{lo}(\tilde{z})\rceil\big) \;\wedge \\
& \Box\begin{pmatrix} \lceil \mathrm{lo}(\tilde{z})\rceil & \Rightarrow & \lceil \neg z\rceil \;\wedge \\ \lceil \mathrm{hi}(\tilde{z})\rceil & \Rightarrow & \lceil z\rceil \;\wedge \\ (\neg(\lceil \mathrm{lo}(\tilde{z})\rceil \vee \lceil \mathrm{hi}(\tilde{z})\rceil) \;\wedge\; (\lceil \neg z\rceil \vee \lceil z\rceil)) & \Rightarrow & l < t^* \end{pmatrix} \\[1ex]
\models\ & \{\text{Law } 11.3.\} \\
& \big((\lceil \neg x\rceil^{t_{\mathrm{nand}}} \vee \lceil \neg y\rceil^{t_{\mathrm{nand}}}) \leadsto \lceil z\rceil\big) \;\wedge\; \big(\lceil x \wedge y\rceil^{t_{\mathrm{nand}}} \leadsto \lceil \neg z\rceil\big) \\[1ex]
\models\ & \{t \geq t_{\mathrm{nand}},\ \text{law } 11.4.\} \\
& \big((\lceil \neg x\rceil^{t} \vee \lceil \neg y\rceil^{t}) \leadsto \lceil z\rceil\big) \;\wedge\; \big(\lceil x \wedge y\rceil^{t} \leadsto \lceil \neg z\rceil\big) \\[1ex]
\models\ & \{(12.2).\} \\
& \mathcal{DL}[\![\mathtt{nand}_t(x,y,z)]\!] \;.
\end{aligned}$$

Thus $\mathtt{nand}_t(x,y,z)\, I_{\mathit{Circuit}}\, \mathrm{NAND}(\tilde{x},\tilde{y},\tilde{z})$ holds, which discharges proof obligation (*iv*) for the structural rule (12.15). ▟

Sensors and actuators. As both $t_{\mathrm{sens}} \geq t^*$ and $t_{\mathrm{act}} \geq t^*$, proof obligation (*iv*) is discharged for structural rule (12.16), dealing with sensors, and (12.17), dealing with actuators, in the same way as for rule (12.15). ▟

Circuit composition. Let $c_1\, I_{\mathit{Circuit}}\, \tilde{c}_1$ and $c_2\, I_{\mathit{Circuit}}\, \tilde{c}_2$. We will not deal formally with the case that $c_1 \,\&\, c_2$ is ill-formed, as then the proof obligation is trivially discharged due to $\mathcal{DL}[\![c_1 \,\&\, c_2]\!] = \mathtt{true}$.

$$\begin{aligned}
& \mathcal{CL}[\![\tilde{c}_1 \,\&\, \tilde{c}_2]\!] \wedge \mathit{abs}_{\mathit{Nodes}(c_1 \,\&\, c_2)} \\[1ex]
\models\ & \{\text{By } (12.11) \text{ and well-formedness, } \mathcal{CL}[\![\tilde{c}_1 \,\&\, \tilde{c}_2]\!] = \mathcal{CL}[\![\tilde{c}_1]\!] \wedge \mathcal{CL}[\![\tilde{c}_2]\!].\} \\
& \mathcal{CL}[\![\tilde{c}_1]\!] \wedge \mathcal{CL}[\![\tilde{c}_2]\!] \wedge \mathit{abs}_{\mathit{Nodes}(c_1 \,\&\, c_2)}
\end{aligned}$$

$$
\begin{array}{ll}
\models & \{abs_{M \cup N} = abs_M \wedge abs_N,\ Nodes(c_1 \,\&\, c_2) = Nodes(c_1) \cup Nodes(c_2).\} \\
& \mathcal{CL}[\![\tilde{c}_1]\!] \wedge \mathcal{CL}[\![\tilde{c}_2]\!] \wedge abs_{Nodes(c_1)} \wedge abs_{Nodes(c_2)} \\[1ex]
\models & \{c_1\, I_{Circuit}\, \tilde{c}_1 \ \text{ and } \ c_2\, I_{Circuit}\, \tilde{c}_2.\} \\
& \mathcal{DL}[\![c_1]\!] \wedge \mathcal{DL}[\![c_2]\!] \\[1ex]
\models & \{(12.3).\} \\
& \mathcal{DL}[\![c_1 \,\&\, c_2]\!] \ .
\end{array}
$$

I.e., $(c_1 \,\&\, c_2)\, I_{Circuit}\, (\tilde{c}_1 \,\&\, \tilde{c}_2)$ holds, which proves obligation (iv) for structural rule (12.18).

Hiding. Let $x \in Node$ and $x\, I_{Node}\, \tilde{x}$ and $c\, I_{Circuit}\, \tilde{c}$. Then

$$
\begin{array}{ll}
& \mathcal{CL}[\![\text{HIDE } \tilde{x} \text{ IN } (\tilde{c})]\!] \wedge abs_{Nodes(\texttt{hide } x \texttt{ in } (c))} \\[1ex]
\models & \{Nodes(\texttt{hide } x \texttt{ in } (c)) = Nodes(c) \setminus \{x\}.\} \\
& \mathcal{CL}[\![\text{HIDE } \tilde{x} \text{ IN } (\tilde{c})]\!] \wedge abs_{Nodes(c)\setminus\{x\}} \\[1ex]
\models & \{(12.12).\} \\
& \exists \tilde{x}, I_{\tilde{x}}, C_{\tilde{x}}.\mathcal{CL}[\![c]\!] \wedge abs_{Nodes(c)\setminus\{x\}} \\[1ex]
\models & \{x,\ I_{\tilde{x}}, \text{ and } C_{\tilde{x}} \text{ do not occur free in } abs_{Nodes(c)\setminus\{x\}}.\} \\
& \exists \tilde{x}, I_{\tilde{x}}, C_{\tilde{x}}.\left(\mathcal{CL}[\![c]\!] \wedge abs_{Nodes(c)\setminus\{x\}}\right) \\[1ex]
\models & \{\models \exists x.abs_x.\} \\
& \exists \tilde{x}, I_{\tilde{x}}, C_{\tilde{x}}.\left(\mathcal{CL}[\![c]\!] \wedge abs_{Nodes(c)\setminus\{x\}} \wedge \exists x.abs_x\right) \\[1ex]
\models & \{x \text{ does not occur free in } \mathcal{CL}[\![c]\!] \wedge abs_{Nodes(c)\setminus\{x\}}.\} \\
& \exists x, \tilde{x}, I_{\tilde{x}}, C_{\tilde{x}}.\left(\mathcal{CL}[\![c]\!] \wedge abs_{Nodes(c)\setminus\{x\}} \wedge abs_x\right) \\[1ex]
\models & \{\text{Definition of } abs \text{ on sets of nodes.}\} \\
& \exists x, \tilde{x}, I_{\tilde{x}}, C_{\tilde{x}}.\left(\mathcal{CL}[\![c]\!] \wedge abs_{Nodes(c)\cup\{x\}}\right) \\[1ex]
\models & \{\models abs_{Nodes(c)\cup\{x\}} \Rightarrow abs_{Nodes(c)}.\} \\
& \exists x, \tilde{x}, I_{\tilde{x}}, C_{\tilde{x}}.\left(\mathcal{CL}[\![c]\!] \wedge abs_{Nodes(c)}\right) \\[1ex]
\models & \{c\, I_{Circuit}\, \tilde{c}.\} \\
& \exists x, \tilde{x}, I_{\tilde{x}}, C_{\tilde{x}}.\mathcal{DL}[\![c]\!] \\[1ex]
\models & \{\tilde{x}, I_{\tilde{x}}, C_{\tilde{x}} \text{ do not occur free in } \mathcal{DL}[\![c]\!].\} \\
& \exists x.\mathcal{DL}[\![c]\!] \\[1ex]
\models & \{(12.3).\} \\
& \mathcal{DL}[\![\texttt{hide } x \texttt{ in } (c)]\!] \ .
\end{array}
$$

As this shows $(\texttt{hide}\ x\ \texttt{in}\ (c))\, I_{Circuit}\, (\text{HIDE}\ \tilde{x}\ \text{IN}\ (\tilde{c}))$, verification condition (iv) is satisfied by rule (12.19). ∎

Complete systems

As a correctness criterion for complete systems we simply take semantic entailment, defining

$$\pi\, I_{System}\, \tilde{\pi} \qquad \text{iff} \qquad \mathcal{CL}[\![\tilde{\pi}]\!] \models \mathcal{DL}[\![\pi]\!]$$

for each hardware language system $\pi \in System$ and each complete CMOS system $\tilde{\pi}$. As I_{System} is the subrelation of I that deals with complete systems, there are two proof obligations associated with I_{System}, namely

ii) for each abstract system π and each concrete system $\tilde{\pi}$ with $(\pi, \tilde{\pi}) \in I$ it holds that $\pi\ \texttt{sat}\ s$ implies $\tilde{\pi}\ \texttt{sat}\ s$ for each $s \in S$,

iv) for each operator $o : t_1, \ldots, t_n \to System$ in O and all $\mathcal{L}$-terms $P_1, \ldots, P_n$ of types $t_1, \ldots, t_n$ and all $\tilde{\mathcal{L}}$-terms $Q_1, \ldots, Q_n$ of types $C_{\text{type}}(t_1), \ldots, C_{\text{type}}(t_n)$ the implication

$$(\forall i \in \{1, \ldots, n\} \,.\, (P_i, Q_i) \in I\ ) \Rightarrow (o(P_1, \ldots, P_n), e(Q_1, \ldots, Q_n)) \in I$$

holds for each $(o, e) \in C_{\text{struct}}$.

For proving (iv) for the structural rule (12.20) — the only rule dealing with systems — let c be a circuit such that $\texttt{system}(c)$ is well-formed and let $\tilde{c}$ be a CMOS circuit with $c\, I_{Circuit}\, \tilde{c}$. Again, we do not expand on the trivial case of ill-formed systems. Then

$$\begin{array}{ll}
 & \mathcal{CL}[\![\text{SYSTEM}(\tilde{c})]\!] \\[1ex]
\models & \{\text{Due to (12.13)}, \mathcal{CL}[\![\text{SYSTEM}(\tilde{c})]\!] = \mathcal{CL}[\![\tilde{c}]\!].\} \\
 & \mathcal{CL}[\![\tilde{c}]\!] \\[1ex]
\models & \{abs_{Nodes(c)} = abs_{\emptyset} = \texttt{true} \text{ as } \texttt{system}(c) \text{ is well-formed.}\} \\
 & \mathcal{CL}[\![\tilde{c}]\!] \wedge abs_{Nodes(c)} \\[1ex]
\models & \{c\, I_{Circuit}\, \tilde{c}.\} \\
 & \mathcal{DL}[\![c]\!] \\[1ex]
\models & \{(12.1).\} \\
 & \mathcal{DL}[\![\texttt{system}(c)]\!]\ ,
\end{array}$$

i.e. $(\texttt{system}(c))\, I_{System}\, (\text{SYSTEM}(\tilde{c}))$. ∎

To prove (ii) let π and $\tilde{\pi}$ be systems with $\pi\, I_{System}\, \tilde{\pi}$ and let s be a duration formula with $\pi\ \texttt{sat}\ s$, i.e. (by definition of $\texttt{sat}$) with $\mathcal{DL}[\![\pi]\!] \models s$. Then

$$\begin{array}{ll}
 & \mathcal{CL}[\![\tilde{\pi}]\!] \\[1ex]
\models & \{\pi\, I_{System}\, \tilde{\pi}, \text{ i.e. } \mathcal{CL}[\![\tilde{\pi}]\!] \models \mathcal{DL}[\![\pi]\!].\}
\end{array}$$

$$\mathcal{DL}[\![\pi]\!]$$

$$\models \{\pi \ \mathtt{sat}\ s.\}$$

$$s\ ,$$

which by definition of $\mathtt{sat}$ shows that $\tilde{\pi}\ \mathtt{sat}\ s$. ▟

This completes the proof of lemma 12.1. Thus, implementing the abstract hardware language by CMOS circuits according to the compilation relation C is correct. By soundness of the proof rule we have shown that for any system $\pi \in \mathit{System}$ and any expectation $s \in S$ we can assert that if π satisfies the expectation s and $C(\pi)$ is defined, then $C(\pi)$ satisfies s. Note that since the set S of possible expectations contains *all* formulae of Hybrid Duration Logic, *the implemented device will satisfy* any *duration property that we can assert of the abstract construction*, even though during the abstract construction we may have exploited properties like $\Box(\lceil\,\rceil \vee \lceil x \vee \neg x\rceil)$ for a node x, i.e. the law of excluded middle, that do *not* hold of the implementation technology.

It should be noted that we have been able to implement the reduction from three- to two-valuedness without any loss of precision of the model. In general, soundly collapsing the state space of a description introduces non-determinism, which yields information loss impairing precision. Thus, fewer properties of a system can usually be deduced from a model with smaller state set. While this is obviously true for our discrete-state model with respect to its continuous-state counterpart, this is not the case for the two-valued reduction (12.2) of property (12.8): as both represent basically the *same* logical formula, but once in a strict three-valued and once in two-valued interpretation, the two-valued model allows at least the same logical deductions. Hence, we have been able to implement a drastic reduction in the size of the state-set describing a circuit (n nodes now amount to 2^n instead of 3^n states) at no price!

12.6 Discussion

This chapter was devoted to analyzing the physical adequacy of the widely used instantaneous-transition model of digital systems within embedded control contexts. Section 12.1 shows that this model is indeed unsound due to the well-known problem of synchronization failure encountered with synchronous hardware in presence of unsynchronized inputs. While this does not mean that instantaneous-transition models are useless, as the gains in tractability obtained from adopting them may outweigh the incurred loss in accuracy,[6] we had a look at alternatives: we have defined a simple design-level model of switching circuitry which retains most of the beauty of the well-known switching-theoretic model, yet is soundly applicable in hybrid control contexts, where synchronization failure is unavoidable due to real-time constraints on reactions to possibly unsynchronized inputs. Soundness has been shown constructively by formalizing and formally analyzing a simple model of CMOS technology in Hybrid Duration Logic and then verifying a compilation of a prototypic hardware language being interpreted by our design-level model to CMOS technology.

[6] The loss in accuracy is usually minor, as synchronization failure is a rare event.

Our model is formalized in terms of standard Duration-Calculus-implementables, as proposed by A. P. Ravn [Rav95b], such that it can be seen as a direct migration path from a subset of the DC implementables to CMOS VLSI that does not take the deviation to an idealized, instantaneous-transition device as intermediate step, as it has been done in the ProCoS projects [HHMO+96]. While the embedding of our model into Duration Calculi is particularly economic, it can also be embedded into automata-based settings. Being formalized on discrete states, our model clearly also does instantaneous state changes, albeit of a particular temporal shape: whenever a state change may happen then the corresponding reverse change is also possible for a certain time period. Thus, automata corresponding to our model are automata where each transition has a corresponding backward transition that will become enabled at the same time, and it is only after a certain time of continuous enabledness of those two that the backward transition is withdrawn and the forward transition has to fire.

Note that these forward/backward transition pairs in fact formalize the preset times of latches, a concept that normally requires a less economic formalization that blows up the state space through explicit introduction of meta-stable states. Hence, our model provides a handle for formally dealing with a phenomenon that is often considered to be too costly to be dealt with in mechanic verification.

Beyond these contributions to modelling issues at digital design level, the verification in section 12.5.4 is one of the rare examples of formal methods being applied to verification crossing the boundary between discrete and continuous models of switching circuitry. Such verification problems occur within VLSI implementation when mapping digital systems to networks of transistors. While our verification is done in a prototypic setting that is characterized by simplistic models, using an extremely reduced hardware language and the first-order model of MOS transistor behaviour, it shows that formal models of refinement can deal with such implementation steps. Given that analog-level simulation of VLSI structures, though time-consuming and never exhaustive, is currently indispensable for debugging this implementation step, verification techniques bridging the semantic gap between digital design-level models of hardware and actual VLSI implementations are worth being explored.

12.6.1 Related work on hardware verification

Currently, most work undertaken in the formal methods community on hardware verification deals with digital models of hardware only. The established correctness properties thus do not cross the boundary between discrete and continuous models of switching circuitry. However, extension of formal methods down to the circuit level of VLSI implementation would be desirable, given the high cost of integrated circuit fabrication and the limits of debugging VLSI through analog-level simulation, as R. P. Kurshan and K. L. McMillan point out in [KM91]. Among the groups that took the challenge of bridging the semantic gap between discrete and continuous models of switching circuitry are F. K. Hanna [Han94a], M. Greenstreet & P. Cahoon [GC94], P. R. Stephan & R. K. Brayton [SB93], who all attack the problem of correctness of discrete hardware models, and R. P. Kurshan & K. L. McMillan

[KM91], who perform verification of individual circuits, not whole circuit models, across the discrete-continuous boundary.

In a sense, our work is complementary to that of F. K. Hanna [Han94a], who investigated composition operators and verification conditions for the steady-state behaviour of the transistor level. As we currently deal with the gate level, our analysis of composition operators at the transistor level is not so sophisticated, but it goes beyond [Han94a] by exploring dynamic behaviour and thus being useful also for non-combinational structures and for timing analysis. M. Greenstreet and P. Cahoon in [GC94] analyze the dynamic behaviour of a nine-transistor toggle element using dynamic systems theory. While this particular analysis is performed on a specific circuit, they aim at identifying a general notion of discrete state in the circuit-level behaviour of digital hardware. The notion of state they propose is based on topological features of solutions to the differential equations describing circuit dynamics: in their interpretation, the discrete states are in one-to-one correspondence to the basins of attraction. However, this implicit notion of state has the shortcoming that it is non-compositional as there is no predefined encoding of states that all components have to obey. Threshold-based models of state, as used by Stephan & Brayton, Kurshan & McMillan, and us, do not suffer from this problem, albeit at the price of a much more simplistic analysis of system dynamics. P. R. Stephan and R. K. Brayton in [SB93] claim to have a formal proof that any two-valued model of digital hardware has to be unsound. However, as our verification of such a model in section 12.5.4 shows, this is the consequence of a too restricted notion of sound model. Essentially, they consider only models that are induced by a functional dependency of the abstract (discrete) states on the continuous states, which is then homogeneously applied in any time *instant*. As none of the hardware components is sensitive to point properties over time, the likely motivation for restricting oneself to such models is adoption of the absolute-observer view of system dynamics, which here — once again — complicates matters beyond necessity.

In contrast to the aforementioned lines of research, R. P. Kurshan and K. L. McMillan in [KM91] do not aim at justifying a particular discrete-state abstraction of digital circuitry, but aim at verification of individual, particularly optimized, VLSI structures. They convincingly demonstrate how sound threshold-based abstraction can be used to transform such a validation problem to a discrete-state one that is in the reach of automata-theoretic methods of verification, as advocated by Vardi and Wolper [VW86]. However, these automata-theoretic methods start from the discrete-state abstraction of the circuit dynamics such that the abstraction step itself still requires manual analysis.

Having derived a sound discrete-state abstraction of circuit dynamics in chapter 12, a similar abstraction step can be performed *effectively* within our theory. While this automation surely comes at a price — sometimes, non-standard thresholds or more than three value ranges may be necessary for verifying a particular circuit — it may be beneficial for standard applications. Furthermore, we have been able to implement a drastic reduction in the size of the resulting state set. Kurshan and McMillan assign in their threshold-based models a distinguished state to each of the value ranges induced by the thresholds. With two thresholds (they allow for an arbitrary number), this would amount to three states per node. We have been able to reduce this number to two states per node without any loss of accuracy. This reduction can be valuable when applying automata-theoretic methods of verification, which is the theme of the next chapter.

13

Decidability for Free: Correctness of Gate-Level Designs

After a slight detour to gate-level models of VLSI we will now return to the main theme of the thesis: decidability of implementation correctness. This time, we explore decidability of correctness of gate-level designs with respect to DL specifications. As the known decidability results for Duration Logic require time-wise discrete models (cf. chapter 6), whereas gate-level dynamics of circuits is potentially unsynchronized, one should not expect positive decidability results with respect to full Duration Logic. Therefore, we will concentrate on decidability of correctness of gate-level designs with respect to control requirements formulae from the $\{\lceil P \rceil, l = k\}$-fragment of DL.

Unfortunately, undecidability of the $\{\lceil P \rceil, l = k\}$-fragment [ZHS93, Theorem 11] implies that even this correctness problem is undecidable. But using similar analysis techniques as in the previous chapter, the design-level semantics can — without any loss of accuracy — be adjusted to yield a decidable correctness problem. We will deal with this adjustment of the gate-level model in the present chapter. The difference to the techniques of the previous chapter is that there qualitative reasoning and conventional abstraction techniques, i.e. elimination of detail, have been predominant means of simplifying the model. In the present chapter, we solely use the technique of postulating *additional*, but in a sense redundant, information: new postulates are simply made, but are shown not to allow unsound deductions concerning satisfaction of requirements by complete systems.

In principle, an instance of this procedure has already been encountered in the previous chapter: while equations (12.8) to (12.13) provide an essentially three-valued model $\mathcal{CL}[\![\cdot]\!]$ of gate-level dynamics, the design-level semantics $\mathcal{DL}[\![\cdot]\!]$ provides a two-valued model. Other than that, the formulae defining $\mathcal{CL}[\![\cdot]\!]$ and $\mathcal{DL}[\![\cdot]\!]$ are superficially similar. Hence, the design-level semantics $\mathcal{DL}[\![\cdot]\!]$ can be interpreted as being derived from the low-level model $\mathcal{CL}[\![\cdot]\!]$ by simply *adding the postulate 'tertium non datur'*, i.e. by *postulating that nodes are two-valued.* The verification in section 12.5.4 shows that this additional postulate does not allow invalid deductions concerning satisfaction of requirements by complete systems. I.e., the additional postulate comes for free: the only price that one might ever have to pay for adding a postulate is unsoundness of the resulting model, which does not occur with $\mathcal{DL}[\![\cdot]\!]$ according to section 12.5.4.

It is obvious that the technique of adding postulates which are 'for free' can simplify verification issues: the new postulates provide additional information on the device to be verified which can be exploited within proofs. In the present chapter, we add the postulate that node behaviour is of 3-bounded variability to the design-level model. Section 13.1 is devoted to showing that this postulate is for free, while section 13.2 demonstrates that by adding this postulate the correctness problem of gate-level designs with respect to $\{\lceil P \rceil, l = k\}$-formulae becomes decidable.

13.1 Postulating bounded variability

Adding the postulate that node behaviour is of 3-bounded variability is simply done through adding conjuncts to the predicative semantics $\mathcal{DL}[\![\cdot]\!]$.

Definition 13.1. *The predicative semantics $\mathcal{DL}_3[\![\cdot]\!] : System \cup Circuit \to$ HDL that restricts node behaviour to 3-bounded variability is defined by*

$$\begin{array}{rcl}
\mathcal{DL}_3[\![\mathtt{nand}_t(x,y,z)]\!] & \stackrel{\text{def}}{=} & \mathcal{DL}[\![\mathtt{nand}_t(x,y,z)]\!] \ \wedge \ \mathit{3bounded}_x \ \wedge \ \mathit{3bounded}_y \ \wedge \ \mathit{3bounded}_z \ , \\
\mathcal{DL}_3[\![\mathtt{sens}_t(\phi,x)]\!] & \stackrel{\text{def}}{=} & \mathcal{DL}[\![\mathtt{sens}_t(\phi,x)]\!] \ \wedge \ \mathit{3bounded}_x \ , \\
\mathcal{DL}_3[\![\mathtt{act}_t(x,\phi)]\!] & \stackrel{\text{def}}{=} & \mathcal{DL}[\![\mathtt{act}_t(x,\phi)]\!] \ \wedge \ \mathit{3bounded}_x \ , \\
\mathcal{DL}_3[\![c_1 \,\&\, c_2]\!] & \stackrel{\text{def}}{=} & \begin{cases} \mathcal{DL}_3[\![c_1]\!] \wedge \mathcal{DL}_3[\![c_2]\!], & \textit{if } c_1 \,\&\, c_2 \textit{ is well-formed,} \\ \mathtt{true} & \textit{otherwise,} \end{cases} \\
\mathcal{DL}_3[\![\mathtt{hide}\ x\ \mathtt{in}\ (c)]\!] & \stackrel{\text{def}}{=} & \exists x. \mathcal{DL}_3[\![c]\!] \ , \\
\mathcal{DL}_3[\![\mathtt{system}(c)]\!] & \stackrel{\text{def}}{=} & \begin{cases} \mathcal{DL}_3[\![c]\!], & \textit{if } \mathtt{system}(c) \textit{ is well-formed,} \\ \mathtt{true} & \textit{otherwise,} \end{cases}
\end{array}$$

where

$$\mathit{3bounded}_v \stackrel{\text{def}}{=} \Box \left(\left(\begin{array}{l} \lceil \neg v \rceil \,;\, \lceil v \rceil \,;\, \lceil \neg v \rceil \,;\, \lceil v \rceil \ \vee \\ \lceil v \rceil \,;\, \lceil \neg v \rceil \,;\, \lceil v \rceil \,;\, \lceil \neg v \rceil \end{array} \right) \Rightarrow \ l > 1 \right) \ .$$

A state variable v satisfying the formula $\mathit{3bounded}_v$ has at most three state changes per time unit, i.e. is of 3-bounded variability.

As the new predicative semantics $\mathcal{DL}_3[\![\cdot]\!]$ is derived from the gate-level model $\mathcal{DL}[\![\cdot]\!]$ by adding conjuncts occurring in positive contexts, it is obvious that $\mathcal{DL}_3[\![\pi]\!] \models \mathcal{DL}[\![\pi]\!]$ holds for any gate-level circuit and any gate-level system π. I.e., the new predicative semantics $\mathcal{DL}_3[\![\cdot]\!]$ is apparently more deterministic than the previous gate-level model. Nevertheless, both semantics are equivalent in the following sense:

Theorem 13.2 (Equivalence of $\mathcal{DL}[\![\cdot]\!]$ and $\mathcal{DL}_3[\![\cdot]\!]$). *Let $\pi \in System$ be a system of the hardware language that contains only components with a switching latency of at least 2.*[1] *Let ϕ be an HDL formula. Then $\mathcal{DL}[\![\pi]\!] \models \phi$ iff $\mathcal{DL}_3[\![\pi]\!] \models \phi$, i.e. correctness of control systems coincides with respect to the two gate-level models $\mathcal{DL}[\![\cdot]\!]$ and $\mathcal{DL}_3[\![\cdot]\!]$.*

[1] Note that this can, for a given system π, always be achieved by accordingly adjusting the time unit, as π contains finitely many components.

Note that the theorem does not deal with arbitrary circuits, but with complete systems only, such that the dynamic behaviour of the nodes of the enclosed circuit is hidden.

Proof. As $\mathcal{DL}_3[\![\pi]\!]$ is derived from $\mathcal{DL}[\![\pi]\!]$ by adding some conjuncts to the predicative characterization, $\mathcal{DL}_3[\![\pi]\!] \models \mathcal{DL}[\![\pi]\!]$ holds for each system π. Hence, $\mathcal{DL}[\![\pi]\!] \models \phi$ implies $\mathcal{DL}_3[\![\pi]\!] \models \phi$. The other implication follows from corollary 13.7 below, which states that $\mathcal{DL}[\![\pi]\!] \models \mathcal{DL}_3[\![\pi]\!]$ holds for each system π. ∎

The crucial idea for showing the remaining implication is that only a part of the trajectory significantly contributes to determining the future evolution of a system, namely states that are stable for a longer period than the latency. Hence, one can differentiate between significant time instants in the evolution of a system, namely time instants where a node value is stable for at least the latency or is itself driven by a gate, and insignificant time instants, which are all the remaining. Trajectories which differ only in insignificant time instants are either both possible behaviours of a given circuit, or are both not.

A node n is potentially driving a gate or an actuator iff its value is stable for at least 2 time units, which is the minimum switching latency of components in the circuits considered. Hence, the potentially driving time instants for node n in trajectory tr are those instants that belong to stability phases of the value of n which last for at least 2 time units:

$$\mathrm{driving}_n(tr) \stackrel{\mathrm{def}}{=} \left\{ t \in \mathit{Time} \;\middle|\; \begin{array}{l} \exists\, t' \in \mathit{Time}, b \in \mathbb{B}\,. \\ \left(\begin{array}{c} t \in [t', t'+2] \\ \wedge\ \overset{\circ}{\forall}\ t \in [t', t'+2]\,.\, tr(t)(n) = b \end{array} \right) \end{array} \right\} .$$

n is driven by a gate or a sensor iff the inputs of the gate or sensor have been stable for at least the latency. Hence, the time instants that n is driven in depend on the circuit structure, which is taken care of by the following inductive definition:

$$\mathrm{driven}_{n,\mathtt{nand}_{\bar{t}}(x,y,z)}(tr) \stackrel{\mathrm{def}}{=} \begin{cases} \emptyset, & \text{if } v \neq z, \\ \left\{ t \in \mathit{Time} \;\middle|\; \begin{array}{l} t \geq \bar{t}\ \wedge \\ \exists\, b \in \mathbb{B}\,. \\ \quad \overset{\circ}{\forall}\ t' \in [t-\bar{t}, t]\,. \\ \qquad \big([\![x \wedge y]\!](tr)(t') = b\big) \end{array} \right\}, & \text{if } v = z, \end{cases}$$

$$\mathrm{driven}_{n,\mathtt{sens}_{\bar{t}}(\phi,x)}(tr) \stackrel{\mathrm{def}}{=} \begin{cases} \emptyset, & \text{if } v \neq x, \\ \left\{ t \in \mathit{Time} \;\middle|\; \begin{array}{l} t \geq \bar{t}\ \wedge \\ \exists\, b \in \mathbb{B}\,.\, \overset{\circ}{\forall}\ t' \in [t-\bar{t}, t]\,. \\ \qquad \big([\![\phi]\!](tr)(t') = b\big) \end{array} \right\}, & \text{if } v = x, \end{cases}$$

$$\mathrm{driven}_{n,\mathtt{act}_{\bar{t}}(x,\phi)}(tr) \stackrel{\mathrm{def}}{=} \emptyset\ ,$$

$$\mathrm{driven}_{n,C_1\,\&\,C_2}(tr) \stackrel{\mathrm{def}}{=} \mathrm{driven}_{n,C_1}(tr) \cup \mathrm{driven}_{n,C_2}(tr)\ ,$$

$$\mathrm{driven}_{n,\mathtt{hide}\ x\ \mathtt{in}\ (C_1)}(tr) \stackrel{\mathrm{def}}{=} \begin{cases} \mathrm{driven}_{n,C_1}(tr), & \text{if } v \neq x, \\ \emptyset, & \text{if } v = x. \end{cases}$$

As already noted, only the significant time instants of a trajectory determine whether a trajectory is a possible behaviour of a circuit or not. This is made precise in the following lemma.

Lemma 13.3. *Suppose C is a circuit with a minimum component latency of at least 2 and tr and tr' are trajectories which*

- *coincide on the external states, i.e. $tr(t)\big|_{\mathit{Varname}\setminus\mathit{Node}} = tr'(t)\big|_{\mathit{Varname}\setminus\mathit{Node}}$ for each $t \in \mathit{Time}$, and*
- *coincide on significant states, i.e. for each node n there is a set $N \subseteq \mathit{Time}$ which is a right neighborhood[2] of $\mathrm{driven}_{n,C}(tr')$, a superset of $\mathrm{driving}_n(tr')$, and which satisfies $(t \mapsto tr(t)(n))\big|_N \approx (t \mapsto tr'(t)(n))\big|_N$.*

Then $tr \models \mathcal{DL}[\![C]\!]$ implies $tr' \models \mathcal{DL}[\![C]\!]$.

Proof. Assume that tr and tr' are trajectories with $tr(t)\big|_{\mathit{Varname}\setminus\mathit{Node}} = tr'(t)\big|_{\mathit{Varname}\setminus\mathit{Node}}$ for each $t \in \mathit{Time}$ and, for each node n, $(t \mapsto tr(t)(n))\big|_N \approx (t \mapsto tr'(t)(n))\big|_N$ for some $N \subseteq \mathit{Time}$ which is a right neighborhood of $\mathrm{driven}_{n,C}(tr')$ and a superset of $\mathrm{driving}_n(tr')$. We show the contrapositive of the claimed implication by induction on the structure of C.

We start with the base case $C = \mathtt{nand}_{\bar{t}}(x,y,z)$. Assume that $tr' \not\models \mathcal{DL}[\![C]\!]$. According to the semantics of HDL and the definition of $\mathcal{DL}[\![\cdot]\!]$ this implies existence of $t \in \mathit{Time}$ with

$$tr', [0,t] \not\models \Big((\lceil\neg x\rceil^{\bar{t}} \vee \lceil\neg y\rceil^{\bar{t}}) \leadsto \lceil z\rceil\Big) \wedge \Big(\lceil x \wedge y\rceil^{\bar{t}} \leadsto \lceil\neg z\rceil\Big) \quad .$$

In the sequel, we inspect the case that

$$tr', [0,t] \not\models \lceil x \wedge y\rceil^{\bar{t}} \leadsto \lceil\neg z\rceil \quad . \tag{13.1}$$

The other case, i.e. $tr', [0,t] \not\models (\lceil\neg x\rceil^{\bar{t}} \vee \lceil\neg y\rceil^{\bar{t}}) \leadsto \lceil z\rceil$, is similar. Suppose that (13.1) holds. By definition, $\lceil x \wedge y\rceil^{\bar{t}} \leadsto \lceil\neg z\rceil = \neg\Diamond(\lceil x \wedge y\rceil^{\bar{t}} ; \lceil z\rceil) = \neg(\mathtt{true} ; \lceil x \wedge y\rceil^{\bar{t}} ; \lceil z\rceil ; \mathtt{true})$. Due to the semantics of HDL, (13.1) thus implies that there is $[a,b] \in \mathit{TimeInterval}$ with $tr', [a,b] \models \lceil x \wedge y\rceil^{\bar{t}} ; \lceil z\rceil$, i.e.

$$\begin{aligned} tr', [a, a+\bar{t}\,] &\models \lceil x \wedge y\rceil \quad \text{and} \\ tr', [a+\bar{t}, b] &\models \lceil z\rceil \quad . \end{aligned}$$

As $\bar{t} \geq 2$, this implies that $[a, a+\bar{t}\,] \subseteq \mathrm{driving}_x(tr') \cap \mathrm{driving}_y(tr')$ and $[a+\bar{t}, a+\bar{t}\,] \subseteq \mathrm{driven}_{z,C}(tr')$. By the correspondence of tr and tr' in a right neighborhood of the significant time instants, this implies

$$\begin{aligned} (t \mapsto tr(t)(x))\,\big|_{[a,a+\bar{t}]} &\approx (t \mapsto tr'(t)(x))\,\big|_{[a,a+\bar{t}]} \ , \\ (t \mapsto tr(t)(y))\,\big|_{[a,a+\bar{t}]} &\approx (t \mapsto tr'(t)(y))\,\big|_{[a,a+\bar{t}]} \ , \text{ and} \\ (t \mapsto tr(t)(z))\,\big|_{[a+\bar{t},a+\bar{t}+\varepsilon]} &\approx (t \mapsto tr'(t)(z))\,\big|_{[a+\bar{t},a+\bar{t}+\varepsilon]} \ \text{ for some } \varepsilon > 0 \ , \end{aligned}$$

[2]As usual, a right neighborhood of a set $X \subseteq \mathbb{R}$ is a set $N \subseteq \mathbb{R}$ such that for each $x \in X$ there is $\varepsilon \in \mathbb{R}_{>0}$ with $[x, x+\varepsilon] \subset N$.

which, according to lemma 5.1, implies

$$\begin{aligned} tr, [a, a+\bar{t}\,] &\models \lceil x \wedge y \rceil \quad \text{and} \\ tr, [a+\bar{t}, \min(a+\bar{t}+\varepsilon, b)] &\models \lceil z \rceil \ . \end{aligned}$$

Due to the definition of the leads-to operator $\leadsto$ this implies $tr \not\models \lceil x \wedge y \rceil^{\bar{t}} \leadsto \lceil \neg z \rceil$, which shows that $tr \not\models \mathcal{DL}[\![C]\!]$. Thus, $tr' \not\models \mathcal{DL}[\![C]\!]$ implies $tr \not\models \mathcal{DL}[\![C]\!]$ for $C = \mathtt{nand}_{\bar{t}}(x, y, z)$ with $\bar{t} \geq 2$.

The other base cases, namely $C = \mathtt{sens}_{\bar{t}}(\phi, x)$ and $C = \mathtt{act}_{\bar{t}}(x, \phi)$ are similar. For the induction step assume that $tr' \not\models \mathcal{DL}[\![C_i]\!]$ implies $tr \not\models \mathcal{DL}[\![C_i]\!]$ for the circuits C_1 and C_2.

Suppose $C = C_1 \,\&\, C_2$. Then

$$\begin{aligned}
& tr' \not\models \mathcal{DL}[\![C_1 \,\&\, C_2]\!] \\
\Leftrightarrow\ & tr' \not\models \mathcal{DL}[\![C_1]\!] \vee \mathcal{DL}[\![C_2]\!] && [\text{Definition of } \mathcal{DL}[\![\cdot]\!]] \\
\Leftrightarrow\ & tr' \not\models \mathcal{DL}[\![C_1]\!] \ \text{ or } \ tr' \not\models \mathcal{DL}[\![C_2]\!] && [\text{Semantics of HDL}] \\
\Rightarrow\ & tr \not\models \mathcal{DL}[\![C_1]\!] \ \text{ or } \ tr \not\models \mathcal{DL}[\![C_2]\!] && [\text{Assumption on } C_i] \\
\Leftrightarrow\ & tr \not\models \mathcal{DL}[\![C_1]\!] \vee \mathcal{DL}[\![C_2]\!] && [\text{Semantics of HDL}] \\
\Leftrightarrow\ & tr \not\models \mathcal{DL}[\![C_1 \,\&\, C_2]\!] && [\text{Definition of } \mathcal{DL}[\![\cdot]\!]]
\end{aligned}$$

which shows the proposition for circuits of the form $C_1 \,\&\, C_2$. The other case of the induction step is that C is of the form $\mathtt{hide}\ n\ \mathtt{in}\ (C_1)$. Then

$$\begin{aligned}
& tr' \not\models \mathcal{DL}[\![\mathtt{hide}\ x\ \mathtt{in}\ (C_1)]\!] \\
\Leftrightarrow\ & tr' \not\models \exists\, n \,.\, \mathcal{DL}[\![C_1]\!] && [\text{Definition of } \mathcal{DL}[\![\cdot]\!]] \\
\Leftrightarrow\ & \exists\, t \in \mathit{Time} \,.\, \Big(tr', [0,t] \not\models \exists\, n \,.\, \mathcal{DL}[\![C_1]\!]\Big) && [\text{Definition of satisfaction}] \\
\Leftrightarrow\ & \exists\, t \in \mathit{Time}, \widetilde{tr'} \in \mathit{Traj} \,. && [\text{Semantics of HDL}] \\
& \quad \begin{pmatrix} t \mapsto \widetilde{tr'}(t)\big|_{\mathit{Varname}\setminus\{n\}} = t \mapsto tr'(t)\big|_{\mathit{Varname}\setminus\{n\}} \wedge \\ \widetilde{tr'}, [0,t] \not\models \mathcal{DL}[\![C_1]\!] \end{pmatrix} \\
\Leftrightarrow\ & \exists\, \widetilde{tr'} \in \mathit{Traj} \,. && [\text{Definition of satisfaction}] \\
& \quad \begin{pmatrix} t \mapsto \widetilde{tr'}(t)\big|_{\mathit{Varname}\setminus\{n\}} = t \mapsto tr'(t)\big|_{\mathit{Varname}\setminus\{n\}} \wedge \\ \widetilde{tr'} \not\models \mathcal{DL}[\![C_1]\!] \end{pmatrix}
\end{aligned}$$

Given such a trajectory $\widetilde{tr'}$, a corresponding trajectory $\widetilde{tr}$ is defined by

$$\widetilde{tr}(t) \ \stackrel{\text{def}}{=}\ \begin{cases} tr(t)(v), & \text{if } v \neq n, \\ \widetilde{tr'}(t)(v), & \text{if } v = n. \end{cases}$$

As tr and tr' are trajectories with $tr(t)\big|_{Varname \setminus Node} = tr'(t)\big|_{Varname \setminus Node}$ for each $t \in Time$ and, for each node n, $(t \mapsto tr(t)(n))\big|_N \approx (t \mapsto tr'(t)(n))\big|_N$ for some set N which is a right neighborhood of $\text{driven}_{n,C}(tr')$ and a superset of $\text{driving}_n(tr')$, the same relationship applies to $\widetilde{tr}$ and $\widetilde{tr'}$: $\widetilde{tr}(t)\big|_{Varname \setminus Node} = \widetilde{tr'}(t)\big|_{Varname \setminus Node}$ for each $t \in Time$ and $(t \mapsto \widetilde{tr}(t)(n))\big|_M \approx (t \mapsto \widetilde{tr'}(t)(n))\big|_M$ for each node n on some set M which is a right neighborhood of $\text{driven}_{n,C}(\widetilde{tr'})$ and a superset of $\text{driving}_n(\widetilde{tr'})$. The induction hypothesis is thus applicable to C_1 interpreted by $\widetilde{tr}$ and $\widetilde{tr'}$. Above derivation can therefore be extended by

$$\begin{aligned}
& \exists\, \widetilde{tr'} \in Traj\,.\\
&\quad \left(\begin{array}{l} t \mapsto \widetilde{tr'}(t)\big|_{Varname \setminus \{n\}} = t \mapsto tr'(t)\big|_{Varname \setminus \{n\}} \;\wedge \\ \widetilde{tr'} \not\models \mathcal{DL}[\![C_1]\!] \end{array} \right) \\
\Rightarrow\; & \exists\, \widetilde{tr} \in Traj\,. && \text{[Induction hypothesis]}\\
&\quad \left(\begin{array}{l} t \mapsto \widetilde{tr}(t)\big|_{Varname \setminus \{n\}} = t \mapsto tr(t)\big|_{Varname \setminus \{n\}} \;\wedge \\ \widetilde{tr} \not\models \mathcal{DL}[\![C_1]\!] \end{array} \right) \\
\Leftrightarrow\; & \exists\, t \in Time, \widetilde{tr} \in Traj\,. && \text{[Definition of satisfaction]}\\
&\quad \left(\begin{array}{l} t \mapsto \widetilde{tr}(t)\big|_{Varname \setminus \{n\}} = t \mapsto tr(t)\big|_{Varname \setminus \{n\}} \;\wedge \\ \widetilde{tr}, [0,t] \not\models \mathcal{DL}[\![C_1]\!] \end{array} \right) \\
\Leftrightarrow\; & \exists\, t \in Time\,.\, \left(tr, [0,t] \not\models \exists\, n\,.\, \mathcal{DL}[\![C_1]\!] \right) && \text{[Semantics of HDL]}\\
\Leftrightarrow\; & tr \not\models \exists\, n\,.\, \mathcal{DL}[\![C_1]\!] && \text{[Definition of satisfaction]}\\
\Leftrightarrow\; & tr \not\models \mathcal{DL}[\![\texttt{hide}\ x\ \texttt{in}\ (C_1)]\!] && \text{[Definition of } \mathcal{DL}[\![\cdot]\!]\text{]}
\end{aligned}$$

This proves that $tr' \not\models \mathcal{DL}[\![\texttt{hide}\ n\ \texttt{in}\ (C_1)]\!]$ implies $tr \not\models \mathcal{DL}[\![\texttt{hide}\ n\ \texttt{in}\ (C_1)]\!]$, which completes the proof of the last case of the lemma. ∎

According to this lemma, satisfaction of the predicative semantics $\mathcal{DL}[\![C]\!]$ of a circuit C remains unaffected if trajectories are modified in insignificant phases. The following lemma shows that this provides enough freedom for finding 3-bounded models of circuit behaviour: for any trajectory tr there is a corresponding trajectory with 3-bounded node behaviour that corresponds to tr on the external states and in all significant time instants.

Lemma 13.4. *Suppose C is a circuit and tr an arbitrary trajectory. Then there is a trajectory tr' which*

1. *coincides with tr on the external states, i.e. $tr(t)\big|_{Varname \setminus Node} = tr'(t)\big|_{Varname \setminus Node}$ for each $t \in Time$,*
2. *satisfies, for each node n, $(t \mapsto tr(t)(n))\big|_N \approx (t \mapsto tr'(t)(n))\big|_N$ for some right neighborhood N of $\text{driven}_{n,C}(tr')$,*

3. *satisfies, for each node n,* $(t \mapsto tr(t)(n))\big|_{\mathrm{driving}_n(tr')} \approx (t \mapsto tr'(t)(n))\big|_{\mathrm{driving}_n(tr')}$, *and*

4. *satisfies* $tr' \models \mathit{3bounded}_n$ *for each node* n.

Proof. Let C be a circuit and tr a trajectory. In order to satisfy property (1.), we define tr' on external states $v \in \mathit{Varname} \setminus \mathit{Node}$ by $tr'(t)(v) \overset{\mathrm{def}}{=} tr(t)(v)$ for each $t \in \mathit{Time}$. For each node $n \in \mathit{Node}$, figure 13.1 on the next page shows how a trajectory $\widetilde{tr}_n$ can be constructed that satisfies $\mathit{3bounded}_n$ and $(t \mapsto tr(t)(n))\big|_N \approx (t \mapsto \widetilde{tr}_n(t)(n))\big|_N$ for some right neighborhood N of $\mathrm{driven}_{n,C}(\widetilde{tr}_n)$. As $t \mapsto \widetilde{tr}_n(t)(n)$ is by construction instable outside N, node n is not driving outside N, yielding $\mathrm{driving}_n(\widetilde{tr}_n) \subseteq N$. Consequently, $(t \mapsto tr(t)(n))\big|_N \approx (t \mapsto \widetilde{tr}_n(t)(n))\big|_N$ implies $(t \mapsto tr(t)(n))\big|_{\mathrm{driving}_n(\widetilde{tr}_n)} \approx (t \mapsto \widetilde{tr}_n(t)(n))\big|_{\mathrm{driving}_n(\widetilde{tr}_n)}$. Thus, defining $t \mapsto tr'(t)(n) \overset{\mathrm{def}}{=} t \mapsto \widetilde{tr}_n(t)(n)$ for each node $n \in \mathit{Node}$, we obtain a trajectory tr' that also satisfies properties (2.) to (4.). ∎

A direct consequence of lemmata 13.3 and 13.4 is that any behaviour of a circuit has a corresponding behaviour which coincides on the external states, yet is internally 3-bounded.

Corollary 13.5. *For each circuit C with a minimum component latency of at least 2 and each trajectory tr with $tr \models \mathcal{DL}[\![C]\!]$ there is a corresponding trajectory tr' with $tr' \models \mathcal{DL}[\![C]\!]$ that coincides with tr on the external states and satisfies $tr' \models \mathit{3bounded}_n$ for each node n of C.*

Proof. Immediate from lemmata 13.3 and 13.4. ∎

Consequently, any trajectory satisfying the predicative semantics $\mathcal{DL}[\![C]\!]$ of a circuit C has a corresponding trajectory satisfying $\mathcal{DL}_3[\![C]\!]$.

Corollary 13.6. *For each circuit C with a minimum component latency of at least 2 and each trajectory tr with $tr \models \mathcal{DL}[\![C]\!]$ there is a corresponding trajectory tr' with $tr' \models \mathcal{DL}_3[\![C]\!]$ that coincides with tr on the external states.* ∎

As systems are circuits where all node values are hidden from the environment, this implies that $\mathcal{DL}[\![\cdot]\!]$ entails the apparently more deterministic semantics $\mathcal{DL}_3[\![\cdot]\!]$:

Corollary 13.7. $\mathcal{DL}[\![\pi]\!] \models \mathcal{DL}_3[\![\pi]\!]$ *for each system π with a minimum component latency of at least 2.*

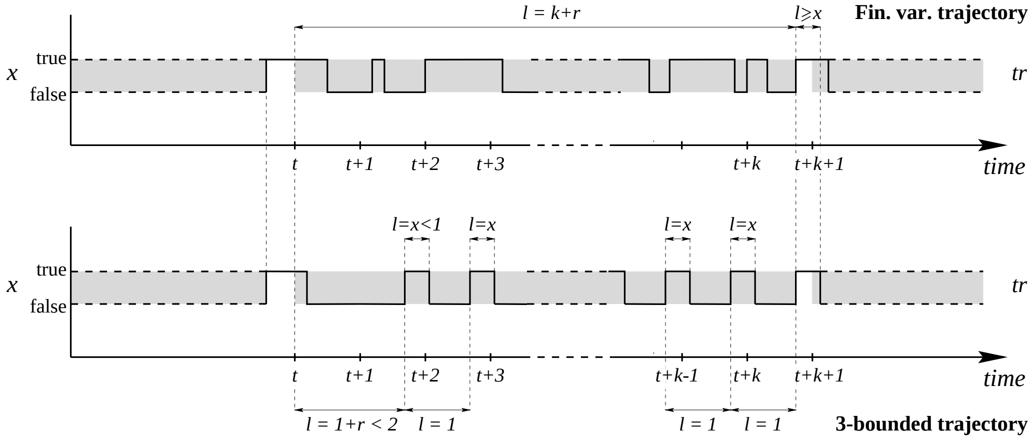
Proof. Suppose π is a system with a minimum component latency of at least 2 and tr a trajectory with $tr \models \mathcal{DL}[\![\pi]\!]$. According to corollary 13.6 there is a corresponding trajectory tr' with $tr' \models \mathcal{DL}_3[\![\pi]\!]$ that coincides with tr on the external states, i.e. $tr(t)\big|_{\mathit{Varname}\setminus\mathit{Node}} = tr'(t)\big|_{\mathit{Varname}\setminus\mathit{Node}}$ for each $t \in \mathit{Time}$. As satisfaction depends only on the valuation of the free variables of a formula and no node state occurs free in $\mathcal{DL}_3[\![\pi]\!]$ because π is a system, this implies $tr \models \mathcal{DL}_3[\![\pi]\!]$. Thus, $tr \models \mathcal{DL}[\![\pi]\!]$ implies $tr \models \mathcal{DL}_3[\![\pi]\!]$, which shows that $\mathcal{DL}[\![\pi]\!] \models \mathcal{DL}_3[\![\pi]\!]$. ∎

If observation of states is limited through a minimum latency of at least 2 then, for any finitely variable trajectory and any node x, a trajectory can be constructed that

- satisfies $\mathit{3bounded}_x$,
- coincides with the finitely variable trajectory within some right neighborhood of all periods where node x is driven, and
- never leaves x stable for 2 or more time units within the remaining period.

This can be shown through the following construction: as the latency is at least 2, the length of the undriven periods (indicated by the shaded parts of the figure) between two driven periods (unshaded parts) is at least 2. In the illustrations it is divided into $l = k + r$ with $k \in \mathbb{N}_{>1}$ and $0 \leq r < 1$. In the 3-bounded trajectory, the first time unit of an undriven phase is filled with a constant signal that depends on the previous state, whereas the remaining $k - 1 + r$ time units are filled by a regular oscillation of frequency 2.

Case 1. States before and after the undriven period coincide:



Case 2. States before and after the undriven period are different:

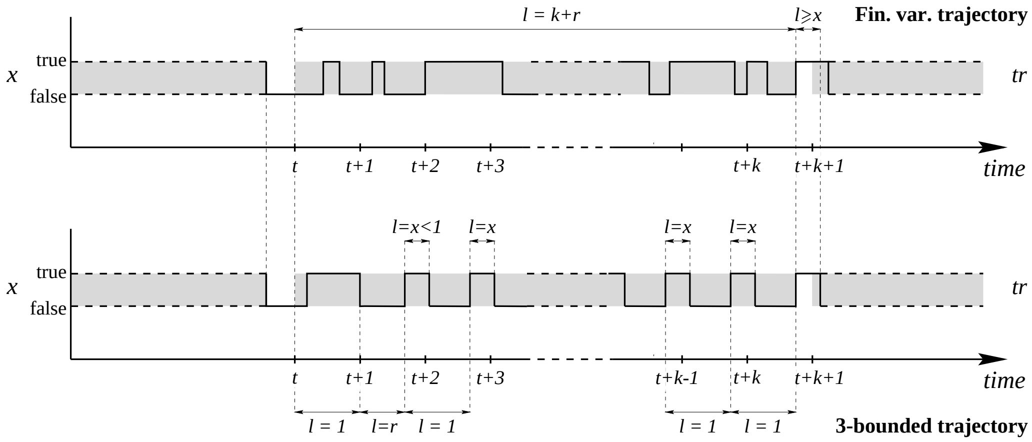


Figure 13.1: Simulation of arbitrary circuit behaviours by 3-bounded behaviours

13.2 Decidability of circuit correctness

The original design-level semantics $\mathcal{DL}[\![\cdot]\!]$ leaves input nodes of circuits completely unconstrained such that lemma 7.2 applies: a circuit C with inputs I satisfies a formula ϕ with $\mathit{free}(\phi) \subseteq I$ iff ϕ is valid. Consequently, undecidability of the validity problem of the $\{\lceil P \rceil, l = k\}$-fragment of DL implies undecidability of circuit correctness with respect to the original design-level semantics:

Lemma 13.8 (Undecidability of circuit correctness). *Circuit correctness with respect to the original design-level semantics $\mathcal{DL}[\![\cdot]\!]$, i.e. the property $\mathcal{DL}[\![C]\!] \models \mathit{Req}$, is in general undecidable for circuits C and formulae Req of the $\{\lceil P \rceil, l = k\}$-fragment of DL.*

Undecidability applies even if the correctness problem is restricted to circuits C and requirements Req with $\mathit{free}(\mathit{Req}) \subseteq \mathit{Nodes}(C)$.

Proof. Consider a circuit C that has inputs I and a requirement Req with $I \supseteq \mathit{free}(\mathit{Req})$. As C has inputs I, which are left unconstrained by the circuit semantics $\mathcal{DL}[\![C]\!]$, validity of Req is a necessary (and obviously sufficient) condition for $\mathcal{DL}[\![C]\!] \models \mathit{Req}$. Hence, deciding $\mathcal{DL}[\![C]\!] \models \mathit{Req}$ amounts to deciding validity of Req, which is in general impossible according to [ZHS93, Theorem 11]. ∎

However, we have shown in the previous section that 3-boundedness of node behaviour may be freely postulated. With this additional assertion, input nodes are no longer unconstrained and hence the proof of lemma 13.8 does not carry over to the new semantics. Indeed, circuit correctness is decidable with respect to the new design-level semantics $\mathcal{DL}_3[\![\cdot]\!]$:

Theorem 13.9 (Decidability of circuit correctness). *Let C be a circuit and Req a formula of the $\{\lceil P \rceil, l = k\}$-fragment of DL with $\mathit{free}(\mathit{Req}) \subseteq \mathit{Nodes}(C)$. It is decidable whether $\mathcal{DL}_3[\![C]\!] \models \mathit{Req}$.*

Proof. Let $n \stackrel{\text{def}}{=} 3 \cdot |\mathit{Nodes}(C)|$. Due to 3-boundedness of node behaviour in $\mathcal{DL}_3[\![C]\!]$, the mapping $t \in \mathit{Time} \mapsto \mathit{tr}(t)\big|_{\mathit{Nodes}(C)}$ is of n-bounded variability for any model tr of $\mathcal{DL}_3[\![C]\!]$. Furthermore, the set of finite models of $\mathcal{DL}_3[\![C]\!]$ is prefix-closed, i.e.

$$\mathit{tr}, [0,e] \models \mathcal{DL}_3[\![C]\!] \quad \text{and} \quad e' \leq e \quad \text{implies} \quad \mathit{tr}, [0,e'] \models \mathcal{DL}_3[\![C]\!] \ ,$$

and extendible, i.e.

$$\mathit{tr}, [0,e] \models \mathcal{DL}_3[\![C]\!] \ \text{ implies } \ \exists\, \mathit{tr}' \in \mathit{Traj}, e' > e \,.\, \begin{pmatrix} \mathit{tr}\big|_{[0,e]} = \mathit{tr}'\big|_{[0,e]} \ \wedge \\ \mathit{tr}', [0,e'] \models \mathcal{DL}_3[\![C]\!] \end{pmatrix} \ .$$

We can thus deduce

$$\begin{aligned} & \mathcal{DL}_3[\![C]\!] \not\models \mathit{Req} \\ \iff \ & \exists\, \mathit{tr} \in \mathit{Traj} \,.\, \begin{pmatrix} \forall\, e \in \mathit{Time} \,.\, \big(\mathit{tr}, [0,e] \models \mathcal{DL}_3[\![C]\!]\big) \ \wedge \\ \exists\, e \in \mathit{Time} \,.\, \big(\mathit{tr}, [0,e] \models \neg \mathit{Req}\big) \end{pmatrix} && \text{[Semantics of DL]} \end{aligned}$$

$$\iff \exists\, tr \in \mathit{Traj}, e \in \mathit{Time}\,.\, \begin{pmatrix} (tr, [0,e] \models \mathcal{DL}_3[\![C]\!]) \;\wedge \\ (tr, [0,e] \models \neg \mathit{Req}) \end{pmatrix} \quad [\mathcal{DL}_3[\![C]\!] \text{ pref.-cl., extendible}]$$

$$\iff \exists\, tr \in \mathit{Traj}, e \in \mathit{Time}\,.\, \begin{pmatrix} t \mapsto tr(t)\big|_{\mathit{Nodes}(C)} \text{ is } n\text{-bounded} \wedge \\ (tr, [0,e] \models \mathcal{DL}_3[\![C]\!]) \;\wedge \\ (tr, [0,e] \models \neg \mathit{Req}) \end{pmatrix}$$

[Node values are 3-bounded]

$$\iff \exists\, tr \in \mathit{Traj}_n, e \in \mathit{Time}\,.\, \begin{pmatrix} (tr, [0,e] \models \mathcal{DL}_3[\![C]\!]) \;\wedge \\ (tr, [0,e] \models \neg \mathit{Req}) \end{pmatrix}$$

[Satisfaction depends only on the valuation of the free variables, $\mathit{free}(\mathit{Req}) \subseteq \mathit{Nodes}(C)$]

$$\iff \exists\, tr \in \mathit{Traj}_n, e \in \mathit{Time}\,.\, \big(tr, [0,e] \models \mathcal{DL}_3[\![C]\!] \wedge \neg \mathit{Req}\big) \qquad [\text{Semantics of DL}]$$

$$\iff \exists\, tr \in \mathit{Traj}_n, e \in \mathit{Time}\,.\, \big(tr, [0,e] \not\models \neg\mathcal{DL}_3[\![C]\!] \vee \mathit{Req}\big) \qquad [\text{Semantics of DL}]$$

$$\iff (\neg\mathcal{DL}_3[\![C]\!] \vee \mathit{Req}) \text{ is invalid over } n\text{-bounded models.}$$

Because of this equivalence, it remains to be shown that validity of $\neg\mathcal{DL}_3[\![C]\!] \vee \mathit{Req}$ over n-bounded models is decidable.

$\neg\mathcal{DL}_3[\![C]\!] \vee \mathit{Req}$ is a formula of the $\{\lceil P \rceil, l = k\}$-fragment of DL, extended by existential quantification. The existential quantifiers, which are introduced by hiding of node values, can be removed by renaming. Therefore note that a formula $\phi \vee \neg\exists\, x\,.\,\big(\mathit{3bounded}_x \wedge \psi\big)$, where x does not occur free in ϕ, is invalid over n-bounded models iff $\phi \vee \neg(\mathit{3bounded}_x \wedge \psi)$ is invalid over n-bounded models:

$$\phi \vee \neg\exists\, x\,.\,\big(\mathit{3bounded}_x \wedge \psi\big) \quad \text{is invalid over } n\text{-bounded models}$$

$$\iff \exists\, tr \in \mathit{Traj}_n, e \in \mathit{Time}\,.\, \Big(tr, [0,e] \not\models \phi \vee \neg\exists\, x\,.\,\big(\mathit{3bounded}_x \wedge \psi\big)\Big)$$

[Def. of validity over n-bounded models]

$$\iff \exists\, tr \in \mathit{Traj}_n, e \in \mathit{Time}\,.\, \begin{pmatrix} tr, [0,e] \models \neg\phi & \wedge \\ tr, [0,e] \models \exists\, x\,.\,\big(\mathit{3bounded}_x \wedge \psi\big) & \end{pmatrix} \qquad [\text{Semantics of HDL}]$$

$$\iff \exists\, tr \in \mathit{Traj}_n, e \in \mathit{Time}\,.\, \begin{pmatrix} tr, [0,e] \models \neg\phi \hfill \wedge \\ \exists\, tr' \in \mathit{Traj}\,. \\ \quad \begin{pmatrix} tr', [0,e] \models \mathit{3bounded}_x \wedge \psi \hfill \wedge \\ \forall\, w \in \mathit{Varname} \setminus \{x\}\,. \\ \quad \forall\, t \in \mathit{Time}\,.\, tr(t)(w) = tr'(t)(w) \end{pmatrix} \end{pmatrix}$$

[Semantics of existential quantification]

$$\iff \exists\, tr \in \mathit{Traj}_n, e \in \mathit{Time}\,.\, \begin{pmatrix} tr, [0,e] \models \neg\phi \hfill \wedge \\ \exists\, tr' \in \mathit{Traj}_n\,. \\ \quad \begin{pmatrix} tr', [0,e] \models \mathit{3bounded}_x \wedge \psi \hfill \wedge \\ \forall\, w \in \mathit{Varname} \setminus \{x\}\,. \\ \quad \forall\, t \in \mathit{Time}\,.\, tr(t)(w) = tr'(t)(w) \end{pmatrix} \end{pmatrix}$$

[$\mathit{3bounded}_x$ implies 3-boundedness of x's behaviour]

$$\iff \exists\, tr \in Traj_n, e \in Time\,.\ \begin{pmatrix} tr,[0,e] \models \neg\phi & \wedge \\ tr,[0,e] \models \mathit{3bounded}_x \wedge \psi \end{pmatrix} \qquad [x \text{ not free in } \phi]$$

$$\iff \exists\, tr \in Traj_n, e \in Time\,.\ \big(tr,[0,e] \not\models \phi \vee \neg(\mathit{3bounded}_x \wedge \psi)\big) \qquad \text{[Semantics of HDL]}$$

$$\iff \phi \vee \neg(\mathit{3bounded}_x \wedge \psi)\ \text{ is invalid over } n\text{-bounded models.}$$

The formula $\neg\mathcal{DL}_3[\![C]\!] \vee Req$ is of the form $\phi \vee \neg\exists\, x\,.\,\big(\mathit{3bounded}_x \wedge \psi\big)$, as $\mathcal{DL}_3[\![C]\!]$ is a conjunction of formulae of the form $\mathit{3bounded}_x \wedge \psi$, some of them existentially quantified. Hence, it can — by uniquely renaming the bound variables and removing the quantifiers — be transformed into a formula $\phi_{C,Req}$ of the $\{\lceil P\rceil, l = k\}$-fragment (without quantifiers) that is valid on n-bounded models iff $\neg\mathcal{DL}_3[\![C]\!] \vee Req$ is valid on n-bounded models.

As validity of formulae of the $\{\lceil P\rceil, l = k\}$-fragment over n-bounded models is decidable according to theorem 6.29, validity of $\phi_{C,Req}$ and thus of $\neg\mathcal{DL}_3[\![C]\!] \vee Req$ over n-bounded models is decidable. Hence, $\mathcal{DL}_3[\![C]\!] \models Req$ is decidable for circuits C and formulae Req of the $\{\lceil P\rceil, l = k\}$-fragment with $free(Req) \subseteq Nodes(C)$. ∎

Comparing theorem 13.9 to lemma 13.8, one can see that adding the postulate that node behaviour is of 3-bounded variability provides a decidable variant of a previously undecidable problem. Yet, as this postulate has been added 'for free', the old and the new model are equivalent in the following strong sense: a *complete system* (including sensors and actuators) is correct within the new model iff it is in the old model; in particular, the postulate does not allow unsound deductions concerning satisfaction of requirements by complete systems. The two models differ only in the encoding of the not directly observable internal dynamics of control circuits: we have exploited the limited observability of internal controller dynamics by the plant for adding an externally unobservable property. This additional property is sufficient for gaining decidability of the *circuit* correctness problem against control requirements from the $\{\lceil P\rceil, l = k\}$-fragment of DL, i.e. for automatically model-checking asynchronous circuits against $\{\lceil P\rceil, l = k\}$-formulae that *do not mention plant states*. This is the correctness problem that remains in a controller design activity once the interfaces have been designed and a specification of the control circuit's desired functionality has been given. It usually accounts for the major part of the verification effort.

Note, furthermore, that the additional postulates do not only make circuit verification much simpler, namely a key-press activity, but also make interface verification not harder: the new model gives all the information about interface circuitry that the old one provides, plus some additional postulates, such that it will never be harder to deduce a property in the new model than in the old model. While the new model allows to verify exactly the *same* control-system/control-problem pairs as the old one, it is always a more convenient tool for doing so!

13.3 Discussion

In the present chapter, we derived a more refined model $\mathcal{DL}_3[\![\cdot]\!]$ of asynchronous circuit behaviour from the design level model $\mathcal{DL}[\![\cdot]\!]$ through postulating that node behaviour is of 3-bounded variability. While hardware engineers to whom the author talked considered the original model $\mathcal{DL}[\![\cdot]\!]$, which describes switching latencies by unpredicatibility of the gate output during switching, a natural abstraction of the continuous-state behaviour, they generally perceived the postulate of 3-boundedness as being dubious and hardly justifiable from device dynamics. Yet we have shown in theorem 13.2 that this postulate can be freely made as it does not allow unsound deduction concerning satisfaction of requirements by complete systems. Therefore, adding this postulate to the semantic model does not impede soundness of the design model. Nevertheless, the additional information provided by the postulate helps setting up a decision procedure for correctness of asynchronous circuits with respect to control requirements from the $\{\lceil P \rceil, l = k\}$-fragment of Duration Logic — indeed, the postulate yields a decidable replacement for an otherwise undecidable correctness problem (lemma 13.8 vs. theorem 13.9).

One might expect that there is a price to be paid for such an extreme reduction of verification effort connected to a certain correctness property, e.g. that other proof obligations within the whole system design become more complicated. But as the new model $\mathcal{DL}_3[\![\cdot]\!]$ provides strictly more information about the target device than the old semantics $\mathcal{DL}[\![\cdot]\!]$, this cannot occur: any proof obligation that can be discharged with respect to $\mathcal{DL}[\![\cdot]\!]$ can be discharged with respect to $\mathcal{DL}_3[\![\cdot]\!]$ with *at most* the same effort. This may be irritating, as on the other hand it needs nothing more than the old model $\mathcal{DL}[\![\cdot]\!]$ for justifying the additional postulate — i.e. the postulate is in a sense already contained in the old model, yet in a form that cannot be exploited if refinement is taken to be trajectory inclusion. Hence, the decidability that has been gained for free is a stunning demonstration of the hypothesis that a particular formalization can cause undecidability even if it is not inherent to the design task.

Given this success of the 'postulates-for-free' technique it should be noted that this technique is by no means unusual in the field of formal models of computation. An important example of this technique is the interleaving assumption found in several concurrency theories: while an execution environment should better not serialize all potentially concurrent actions, it can be designed such that the effect is the same as if they were. If this condition is met it can be *postulated* that actions occur interleaved, and this postulate can be exploited for simplifying the verification task by having less transitions to examine. This simplification is for free, provided that the execution mechanism carefully controls the interaction of concurrent actions. Otherwise, however, the interleaving assumption may be harmful, as it can give rise to unsound deductions about the actual program behaviour.

Another postulate frequently encountered in concurrency theory is that certain actions be atomic. Again, such an assumption will hardly ever be true with respect to a detailed physical analysis of the underlying execution mechanism. Yet it can often be justified by semantic independence properties between the possible actions of a system (for a discussion of semantic independence and its relationship to atomicity cf. e.g. [BL89]). Semantic independence serves as a tool for showing that the postulated atomicity properties are for free in that it shows that the postulates never allow unsound deductions concerning the observable system behaviour. In fact, the analysis techniques for observability constraints presented in chapter 10 evolved from earlier work of the author on semantic independence of imperative program fragments [FSW95].

These examples show that postulates that are obtained from exploiting observability constraints have proved extremely useful in the field of qualitative-time concurrency theories. The author believes that similar analyses of observability constraints in real-time embedded control, as commenced in chapter 12, may contribute to the tractability of formal methods for real-time controller design. The findings of the present chapter provide some evidence for this claim, yet are only a start: physical bounds to information transport, be it noise, speed of signals, or drift and granularity of clocks, are ubiquitous in control systems. As they are yet often not reflected in the computational models of real-time systems, a bunch of new results waits to be discovered!

14

Conclusions

The primary goal of this thesis is to show that *simplified* models of system dynamics, as often used in formal methods dealing with embedded control, actually may *complicate* application to design problems. This hypothesis was drawn from the observation that some of the formal models can encode more information in a timed behaviour than information-theoretically possible for embedded control, such that some of the intricacy of designing embedded controllers within these formalisms might be an artefact of the formalism rather than an inherent complexity of the design task.

We have given some weight to this hypothesis by demonstrating it on the core of the Duration Calculi, which is a major group of calculi proposed for the rigorous development of embedded real-time control systems [Zho93]. Albeit some interesting new techniques for the Duration Calculi have been developed along this course, e.g. synthesis and decision procedures, we claim that the impact of the findings goes much deeper. Formal methods are intended to be mathematically rigorous system design instruments, aiming at subjecting the development process to mathematical verification. Hence, the only rationale for taking idealizations in the formal representation of a design problem is that the gains in tractability of the problem may outweigh the loss of accuracy. However, if such semantic idealizations complicate rather than simplify the design problem then there is no justification for taking them. Thus, our results may be interpreted as an indication that liberating formal models of embedded control from some of the idealizations that have been imposed for smoothening the models may enhance *both* their accuracy and their tractability.

As this implies that similar analyses of other formal models of embedded-system specification and implementation may yield similar gains, there is quite some potential for extending our results to other temporal logics or even for exploiting them in hardware synthesis technologies. We will discuss these issues after summarizing the direct results of the thesis.

14.1 Summary of results

The hypothesis that some of the intricacy of designing embedded controllers with formal methods might be a by-product of the formalization rather than inherent to the design task was in fact derived from a quite formal insight which is explained in chapter 2: it was observed that a major number of undecidability results for real-time and hybrid-systems formalisms relies on encoding infinite information in compact Cartesian products of state and time, whereas only a finite part thereof can become effective in any embedded control application due to the finite capacity of sensor and actuator channels. This means that the physical interpretation of these undecidability results in the field of embedded control is unclear.

Although above observation shows that the according formalisms are representations of the design problem that are biased by unnecessary detail, it does neither reveal what kind of detail should better be removed nor whether reasoning techniques can provide an isolation from the extraneous detail. We decided to investigate these questions on a testbed and selected Duration Logic (defined in chapter 5), the core of the Duration Calculi [Zho93]. Duration Logic is a metric dense-time interval temporal logic that is tailored towards reasoning about durational constraints on time-dependent Boolean-valued states. This provides a logic with a very flexible vocabulary for specifying and reasoning about embedded real-time systems, albeit at the price of undecidability of validity of formulae [ZHS93]. In chapter 6, we have expanded on these undecidability results, thereby — for the first time ever — also giving a positive decidability result for a true Duration Calculus with unrestricted negation on a dense time domain (theorem 6.24). However, this decidability result depends on the drastic restriction of the model class to time-wise discrete trajectories, i.e. trajectories where state changes occur only in integer time instants. This model class does not even cover the possible behaviours of embedded controllers, which are usually subject to unsynchronized inputs. Therefore, this decidability result alone is insufficient for overcoming the lack of automatic controller verification and synthesis procedures based on Duration Calculi. Indeed, as was shown in section 7.1, there is no remedy against the shortage of *automatic* tool support for the Duration Calculi if the semantic criterion for implementation correctness is trajectory inclusion.

At this point, the observation concerning the lacking physical interpretation of undecidability results dealing with the full model class of finitely variable trajectories can be put to advantage. Sensors and actuators form channels of finite capacity, and thus cannot react to all the small-scale differences between trajectories that are used in the undecidability results for encoding infinite information within compact Cartesian products of state and time. This bottleneck to information flow through sensors and actuators makes actual embedded controllers less sensitive to the environment evolution than the absolute observer underlying a so-called observation-oriented semantics. By explicitly modelling sensor and actuator dynamics, one can analyze and exploit that lacking discriminative power. Through this trick, we obtained a synthesis procedure for synchronous controllers from Duration Logic in chapter 8. As far as we know, this is the first positive result concerning feasibility of automatic synthesis from a *dense-time* Duration Calculus. Previous work within the ProCoS projects[1] had concentrated on automatic synthesis for a *discrete-time* dialect of Duration Calculus [HO93], which is a much more modest problem because discrete-time Duration Calculus is

[1]For a description of the ProCoS projects, the reader is referred to [B^+89, B^+93]. Brief technical overviews can be found in the publications [BFOR93, HHF^+94]; more complete overviews are available from the ProCoS `WWW` archive (URL: `http://www.comlab.ox.ac.uk/archive/procos/procos2.html`) as ProCoS reports [BHH^+94, $HHMO^+96$].

decidable and its model classes correspond to effectively constructible regular ω-languages [SS94, Han94b]. Hence, synthesis from discrete-time Duration Calculus can use classical techniques, whereas synthesis from its dense-time counterpart builds upon physical analysis of the design problem in order to avoid the in general impossible construction of the full model class of a dense-time formula.

As an aside, we would like to remark that the new, dense-time-based, synthesis result indeed fills a gap in the ProCoS design model. While the discrete-time synthesis method was available, it did not interface smoothly to the dense-time design model of ProCoS. The reason is that the discrete-time interpretation of the temporal operators quantifies over a discrete instead of a dense time domain, yielding a significantly different semantics of Duration Calculus than its dense-time counterpart. A straightforward mixture of dense-time and discrete-time reasoning is thus in general unsound. This implies that the choice between these two views of system dynamics has to be done in an early design phase. The dichotomy between dense and discrete-time interpretations is thus a fundamental system-modelling issue rather than a question of mainly theoretical interest. When ProCoS opted for the dense-time view in order to simplify interfacing to continuous analysis of the plant, this was deemed to imply impossibility of automatic model checking and automatic synthesis due to the reasons explained in section 7.1. However, chapter 8 shows that this problem can be overcome by careful analysis of the design problem. This adds, for the first time ever, a significant amount of automation to the ProCoS framework. The gains to be expected are drastic, as our treatment of the gas-burner case study [RRH93] in chapter 9 witnesses.

Instrumental to that breakthrough in automation has been the insight that two common proof patterns of undecidability results for real-time and hybrid-systems formalisms fail to encode inherent complexity of the controller design problem. In chapter 10 we have expanded on the basic principles underlying our first successful exploitation of that observation. As any component in a reactive system is subject to reactivity constraints, every component induces similar equivalences on the trajectories as sensors and actuators do. Thus, a generalization of test preorders [Hen88] qualifies as a natural notion of refinement in embedded systems. Using proof rules for showing that a compilation is correct with respect to this notion of refinement, a design-level semantics of CMOS circuits was shown to be sound with respect to a somewhat simplified continuous model of CMOS dynamics in chapter 12.

Again, such a verification of a discrete-state model of digital hardware with respect to continuous circuit-level models of the VLSI implementation has rarely been undertaken. While section 12.6.1 gives a more complete account of that work, we will here concentrate on those contributions that are closely related to our findings. P. R. Stephan and R. K. Brayton in [SB93] claim to have a formal proof that any two-valued model of digital hardware has to be unsound. However, as our verification of such a model in chapter 12 shows, this is the consequence of a too restricted notion of sound model. Essentially, they consider only models that are induced by a functional dependency of the abstract (discrete) states on the continuous states, which is then homogeneously applied in any time *instant*. As none of the hardware components is sensitive to point properties over time, the likely motivation for restricting oneself to such models is adoption of the absolute-observer view of system dynamics, which here — once again — complicates matters beyond necessity.

R. P. Kurshan and K. L. McMillan in [KM91] do not aim at justifying a particular discrete-state abstraction of digital circuitry, but aim at verification of individual, particularly optimized, VLSI structures. They convincingly demonstrate how sound threshold-based abstraction can be used to transform such a validation problem to a discrete-state one that is in the reach of automata-theoretic methods of verification, as advocated by Vardi and Wolper [VW86]. However, these automata-theoretic methods start from the discrete-state abstraction of the circuit dynamics such that the abstraction step itself still requires manual analysis.

Having derived a sound discrete-state abstraction of circuit dynamics in chapter 12, a similar abstraction step can be performed *effectively* within our theory. Hence, automatic verification of gate-level description of circuits that is sound with respect to a simple SPICE-like circuit-level model is possible whenever refinement of a DL-based specification by the discrete-state semantics of a circuit is decidable. While the latter problem is undecidable even with respect to specifications in the $\{\lceil P \rceil, l = k\}$-fragment of DL, we can once again exploit observability constraints to transform it into a simpler, decidable problem. In chapter 13, we have extended the discrete-state semantics of gate-level circuits by adding a postulate that the internal behaviour of circuits is of bounded variability. While it is (comparatively) easy to see why this postulate yields decidability of the circuit correctness problem, it is not that simple to see why one would be allowed to simply state this postulate. In a nutshell, the rationale is that violation of this postulate would never be observable in any embedded control context due to observability constraints. Hence, nothing wrong can be deduced from this postulate, such that it may be freely added and can be exploited for simplifying the verification task.

Altogether, exploitation of observability constraints in embedded control has thus lead to a new synthesis and a new decidability result for hardware implementations with respect to DL-based control problems. Exploitation of a rather straightforward insight can thus provide automatic procedures for problems that were previously considered to be out-of-reach of mechanization. As this success was obtained by replacing a simplistic notion of refinement, namely trajectory inclusion with respect to 'natural' behavioural models, by more complex ones, it provides evidence to our claim that *simplified* models of system dynamics actually may *complicate* application to design problems.

14.2 Further research

When doing the research leading to this thesis, we have chosen a particular standpoint towards embedded controller design: we have opted for the logic-based approach to system specification, in particular its Duration-Calculus-based facet as advocated by the ProCoS projects. However, this is a very special point of view that fails to provide sufficient expressiveness for requirements capture and controller design in many respects. Other temporal logics offer other primitives for describing system dynamics in their vocabulary, which may at times be more suitable. More fundamentally, temporal logics may in general fail to offer adequate expressiveness for analyzing and documenting the requirements of real embedded-control problems. Analyzing physical, chemical, or biological processes reveals difficult parameter extraction problems such that system dynamics can generally be described only insufficiently. Thus, it is questionable whether the sharp distinction between admissible and faulty controller behaviour that is imposed by the notion of satisfaction of a formula suffices for describing functional requirements, or should better be confined to clearly identified safety constraints.

While the former possible shortcomings of the vocabulary of the Duration Calculi motivate application of our ideas to extended Duration Calculi or to other temporal logics, the latter lead to the problem of integration of our techniques with other controller design concepts: e.g., classical, knowledge-based, fuzzy or neural concepts.

14.2.1 Liveness and Fairness in Duration Calculi

While it has been frequently observed that liveness and fairness are less important concepts in metric time logics than in qualitative time logics, mainly because metric time allows to express time-bounded progress properties as safety properties, their complete lack in Duration Calculus has sometimes been regretted. Among the possible remedies is J.-U. Skakkebæk's proposal in [Ska94] to extend Duration Calculus with Venema's outward chop operators [Ven91], i.e. with modalities that look temporally forward or backward out of the current observation interval by using a chop point that is in the future or the past of the observation interval. An equivalent suggestion has recently been made by P. Pandya, who extended Mean-Value Calculus — a specific Duration Calculus — with weak chop inverses [Pan96].

While both proposals extend Duration Calculus with both backward and forward looking modalities, only the latter is essential for encoding liveness and fairness properties. We conjecture that at least the essential extension of DL with a forward looking modality like 'forward chop' or the weak right inverse of chop[2] will not affect the decidability results of chapter 6. However, as the encoding of the finite models becomes more complicated, the details have to be worked out and the synthesis procedure of chapter 8 and the circuit verification procedure of chapter 13 have to be adjusted.

14.2.2 Application to other temporal logics

This thesis is in its majority devoted to *exemplifying* the consequences of a general observation, namely the missing physical interpretation of certain undecidability results, on a single class of formalisms, namely the Duration Calculi. However, the proof patterns analyzed in section 2.1 are frequently used in the field of dense-time temporal logics. Hence, a lot of other logics could and should be subjected to a similar analysis as DL in chapters 8, 12, and 13 of this thesis.

[2]The *forward chop* operator is the outward looking modality $\rhd$ defined by

$$tr, [b,e] \models \phi_1 \rhd \phi_2 \quad \stackrel{\text{def}}{\iff} \quad \exists\, m \in \mathit{Time}\,.\, \begin{pmatrix} m \geq e & \wedge \\ tr, [b,m] \models \phi_1 \wedge \\ tr, [e,m] \models \phi_2 \end{pmatrix} \quad .$$

The weak right inverse of chop is its dual.

Some of the consequences of such an investigation are immediate: as chapter 13 shows that more than n-bounded trajectories are not worth considering, timed automata can in general be considered to be closed under complementation due to the results of T. Wilke [Wil94b]. Thus, such interesting problems as trajectory inclusion between timed automata or between timed automata and logics that can — as a whole, or just their positive or negative fragments — be effectively mapped to timed automata become decidable. This would provide novel key-press verification methods and, exploiting timed-automata based synthesis, automatic synthesis procedures for temporal logics that currently seem to be out of scope of such techniques.

14.2.3 Non-Boolean requirements

Satisfaction of a logical specification just constitutes a yes/no criterion, which is insufficient for describing a variety of design goals, like performance issues, where not meeting a certain performance does not make the product false, but less versatile. It seems that logical specification is only well-suited for specifying safety constraints (in system-engineering terms, not in the technical sense of temporal logics) of a system such that it covers only a small part of the scope of system analysis, requirements capture, and system development. Combinations with other techniques have to be investigated. It may be promising to constrain usage of logics to just development of safety sub-systems, and to implement performance-relevant control functionality by proven engineering means like fuzzy control or even neural nets. However, means of decomposing system requirements such that they fit to such different design paradigms, as well as techniques for controlling feature interaction between system components built in vastly different technologies, are currently not well-understood, necessitating further research.

14.3 Possible exploitation

If, as Berthold Brecht formulates in his drama 'The Life of Galilei' [Bre39], the ultimate objective of science is to make life easier or better, then every scientific idea should offer some potential for exploitation. Although the theories exposed in this thesis are far from being ripe for industrial application, we can see some such potential:

14.3.1 Automatic hardware synthesis

Due to the ever growing economic significance of compilation and automatic synthesis of hardware from abstract descriptions, a wide range of abstraction levels for the source language of such procedures has recently been studied, reaching from hardware-oriented structure descriptions over algorithmic descriptions by imperative languages (in a broad sense, including process calculi) to temporal logics. The obvious objective of studying more abstract source languages has been to allow the user to say *what* he or she wants the circuit to do rather than *how* to achieve this. As hardware generally engages in an ongoing interaction with its environment, temporal logics, which have a vocabulary specifically tailored towards describing properties of time-dependent activity, lend themselves to such a task. However, there is a diversity of temporal logics with different expressiveness, and only very few of them have currently been examined for their prospects as a source language for automatic hardware synthesis. One primary reason is that logic-based synthesis procedures are commonly derived from decision procedures, which exist only for temporal logics of accordingly restricted expressiveness.

In chapter 8, we have been able to show that contrary to common belief undecidability of Duration Logic does not imply impossibility of automatic synthesis from it. An automatic synthesis method has been developed that — albeit certainly not of immediate practical importance — witnesses that very abstract temporal logics like the Duration Calculi could be used as source languages in hardware synthesis. The implications of this result go far beyond Duration Calculus. Similar techniques should be able to pave the way for hardware synthesis methods based on other temporal logics which currently seem out of scope of automatic synthesis.

This would enable the use of much more abstract programming paradigms than currently used in high-level hardware synthesis, including — as temporal logics feature negation — the very powerful concept of *programming by counterexample*. This programming technique exploits the fact that in safety-critical systems it is often simpler to say what should never happen and how such an undesirable situation may arise than to explicitly formulate a control strategy that avoids it. Using hardware synthesis techniques that support the paradigm of programming by counterexample, it suffices to describe the undesirable situations.

However, effective treatment of programming-by-counterexample requires complementation of automata, the complexity of which may easily become prohibitive. Hence, usage of such technologies should probably be constrained to designing and synthesizing the control paths of hardware devices, in combination with more conventional techniques for describing the data paths.

14.3.2 Automatic verification and automatic synthesis of embedded control software

The statements made with respect to automatic hardware synthesis in the previous section apply to software-controlled embedded systems also, be it microcontrollers or stored-program control. While the time-scale of transactions in such systems is substantially larger than that encountered in VLSI, the basic impact of more abstract programming paradigms is similar. Although up-to-now relatively few incurable failures — but lots of curable ones like wrong invoices etc. — have been doubtlessly traced back to software failures [Hoa96], the need for strict control of software quality becomes increasingly obvious. Hence, the methods sketched in this thesis, be it automatic synthesis or model-checking, could well be helpful for designing the vital, safety-related parts of embedded controllers. However, a thorough basis for combining these methods with standard techniques of controller design would be a prerequisite for this due to the inherent limitations of logical specification discussed in section 14.2.3.

14.3.3 Semantics of high-level hardware description languages

In chapter 12 we have shown how abstract, design-level semantics of hardware devices can be built that soundly represent implementation dynamics, i.e. that are *safe abstractions* rather than approximations of low-level dynamics. This is in contrast to the state of affairs encountered in commercial design languages like VHDL, where some divergence between the deterministic design-level semantics (for technical reasons called the simulation semantics) and the implementation semantics is tolerated. Such semantic inconsistency is certainly a peculiar design decision for a hardware description language, albeit partly justifiable by an engineering pragmatics that uses simulation as an indispensable debugging tool, yet would not accept non-deterministic simulator behaviour.

Formal verification, as able to justify a circuit on *all* possible inputs in a single sweep instead of testing just one of the perhaps infinitely many input sequences per simulation run, now starts rivaling simulation as an effective method of ensuring the correctness of digital systems. The growing success of formal verification in the field is mostly due to the efficient key-press verification techniques offered by symbolic model-checking [McM93]. Such verification techniques can cope with semantic non-determinism easily by just the same means that allow them to effectively deal with the infinite set of possible input sequences. A dissimilarity between design-level semantics and implementation semantics is thus no longer justifiable in such a setting. It is, however, not a simple task to suggest a design semantics that is sufficiently abstract to support a tractable design process and at the same time sound with respect to the implementation semantics. The techniques described in chapter 10 and applied to a simple example in chapter 12 might help in such an enterprise.

Bibliography

[ABL96] J.-R. Abrial, E. Börger, and H. Langmaack, eds. *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*, volume 1165 of *Lecture Notes in Computer Science*. Springer-Verlag, October 1996.

[ACHH93] R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In Grossman et al. [GNRR93], pages 209–229.

[ACM96] E. Asarin, P. Caspi, and O. Maler. A Kleene theorem for timed automata. Unpublished manuscript, to appear as [ACM97]. Verimag, Grenoble, France, April 1996.

[ACM97] ——. A Kleene theorem for timed automata. In G. Winskel, ed., *12th Annual IEEE Symposium on Logic in Computer Science (LICS'97)*. IEEE Computer Society Press, to appear 1997.

[AD94] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[ALN+91] J.-R. Abrial, M. K. O. Lee, D. S. Neilson, P. N. Scharbach, and I. H. Sørensen. The B-method. In S. Prehn and W. J. Toetenel, eds., *VDM'91: Formal Software Development Methods — Vol. 2: Tutorials*, volume 552 of *Lecture Notes in Computer Science*. Springer-Verlag, 1991.

[AMP95] E. Asarin, O. Maler, and A. Pnueli. Symbolic controller synthesis for discrete and timed systems. In P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, eds., *Hybrid Systems II*, volume 999 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995.

[B+89] D. Bjørner et al. A ProCoS project description - ESPRIT BRA 3104. *Bulletin of the European Association for Theoretical Computer Science (EATCS)*, 39:60–73, 1989.

[B+93] J. P. Bowen et al. A ProCoS II project description: ESPRIT Basic Research Project 7071. *Bulletin of the European Association for Theoretical Computer Science (EATCS)*, 50:128–137, June 1993.

[BFOR93] J. P. Bowen, M. Fränzle, E.-R. Olderog, and A. P. Ravn. Developing correct systems. In *Proc. 5th Euromicro Workshop on Real-Time Systems, Oulu, Finland*, pages 176–189. IEEE Computer Society Press, June 1993.

[BHH+94] J. Bowen, C. A. R. Hoare, M. R. Hansen, A. P. Ravn, H. Rischel, E.-R. Olderog, M. Schenke, M. Fränzle, M. Müller-Olm, He Jifeng, and Zheng Jianping. Provably correct systems. ProCoS Technical Report COORD JB 7/1, Oxford University Computing Laboratory, September 1994. Tutorial material for the 1994 Formal Techniques in Real-Time and Fault-Tolerant Systems conference (FTRTFT'94).

[BHL+96] J. Bowen, C. A. R. Hoare, H. Langmaack, E.-R. Olderog, and A. P. Ravn. A ProCoS II project final report: ESPRIT Basic Research Project 7071. *Bulletin of the European Association for Theoretical Computer Science (EATCS)*, 59, June 1996.

[BJ78] D. Bjørner and C. B. Jones. *The Vienna Development Method: The Meta-Language*, volume 61 of *Lecture Notes in Computer Science*. Springer-Verlag, 1978.

[BL69] J. R. Büchi and L. H. Landweber. Solving sequential conditions by finite-state strategies. *Trans. Amer. Math. Soc.*, 138:295–311, 1969.

[BL89] E. Best and C. Lengauer. Semantic independence. *Science of Computer Programming*, 13(1):23–50, December 1989.

[BL95] A. Bouajjani and Y. Lakhnech. Timed automata + temporal logic: Expressiveness and decidability. In I. Lee and S. A. Smolka, eds., *CONCUR '95: Concurrency Theory*, volume 962 of *Lecture Notes in Computer Science*, pages 531–546. Springer-Verlag, 1995.

[BLR95] A. Bouajjani, Y. Lakhnech, and R. Robbana. From duration calculus to linear hybrid automata. In P. Wolper, ed., *Computer Aided Verification (CAV '95)*, volume 939 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995.

[Bre39] B. Brecht. Leben des Galilei. Denmark, 1939. © Suhrkamp Verlag, Berlin, 1955.

[CM73] T. J. Chaney and C. E. Molnar. Anomalous behavior of synchronizer and arbiter circuits. *IEEE Transactions on Computing*, 22:421–422, 1973.

[CW75] G. R. Couranz and D. F. Wann. Theoretical and experimental behavior of synchronizers operating in the metastable region. *IEEE Transactions on Computing*, 24:604–616, 1975.

[Die71] D. L. Dietmeyer. *Logic Design of Digital Circuits*. Allun and Bacon, Inc., 1971.

[DS89] J. Davies and S. Schneider. An introduction to Timed CSP. Technical Monograph 75, Oxford University Computing Laboratory, 1989.

[FMO93] M. Fränzle and M. Müller-Olm. Drift and granularity of time in real-time system implementation. ProCoS Technical Report Kiel MF 10/2, Christian-Albrechts-Universität Kiel, Germany, August 1993.

[Frä94] M. Fränzle. Test preorder and refinement. ProCoS Technical Report Kiel MF 16/2, Christian-Albrechts-Universität Kiel, Germany, December 1994.

[Frä95] ——. A discrete model of VLSI dynamics in hybrid control applications. ProCoS Technical Report Kiel MF 17/3, Christian-Albrechts-Universität Kiel, Germany, April 1995.

[Frä96a] ——. Hardware synthesis from temporal logic: Undecidability need not matter. Position paper, Hardware Synthesis and Verification Workshop, Cornell University, Ithaca, USA, August 1996.

[Frä96b] ——. Synthesizing controllers from duration calculus. In Jonsson and Parrow [JP96], pages 168–187.

[FSW95] M. Fränzle, B. v. Stengel, and A. Wittmüss. A generalized notion of semantic independence. *Information Processing Letters*, 53:5–9, 1995.

[FvK93] M. Fränzle and B. von Karger. Proposal for a programming language core for ProCoS II. ProCoS Technical Report Kiel MF 11/3, Christian-Albrechts-Universität Kiel, Germany, August 1993.

[GC94] M. R. Greenstreet and P. Cahoon. How fast will the flip flop? In *Async '94*. IEEE Computer Society Press, November 1994.

[GD85] L. A. Glasser and D. W. Dobberpuhl. *The Design and Analysis of VLSI Circuits*. Addison-Wesley VLSI Systems Series. Addison-Wesley Publishing Company Inc., 1985.

[GNRR93] R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, eds. *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993.

[Han94a] K. Hanna. Reasoning about real circuits. In T. F. Melham and J. Camilleri, eds., *Higher Order Logic Theorem Proving and its Applications*, volume 859 of *Lecture Notes in Computer Science*. Springer-Verlag, September 1994.

[Han94b] M. R. Hansen. Model-checking discrete duration calculus. *Formal Aspects of Computing*, 6(6A):826–845, 1994.

[Heh84] E. C. R. Hehner. Predicative programming. *Communications of the ACM*, 27:134–151, 1984.

[Hen88] M. Hennessy. *Algebraic Theory of Processes*. MIT Press series in the foundations of computing. The MIT Press, 1988.

[HHF+94] He Jifeng, C. A. R. Hoare, M. Fränzle, M. Müller-Olm, E.-R. Olderog, M. Schenke, M. R. Hansen, A. P. Ravn, and H. Rischel. Provably correct systems. In Langmaack et al. [LdRV94], pages 288–335.

[HHMO+96] He Jifeng, C. A. R. Hoare, M. Müller-Olm, E.-R. Olderog, M. Schenke, M. R. Hansen, and H. Rischel. The ProCoS approach to the design of real-time systems: Linking different formalisms. ProCoS Technical Report DTU APR 25/1, Oxford University Computing Laboratory, Christian-Albrechts-Universität Kiel, Universität Oldenburg, and Technical University of Denmark, February 1996. Tutorial material for the 1996 Formal Methods Europe conference (FME'96).

[HKPV95] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on the Theory of Computing*, pages 373–382. ACM, 1995.

[HMM83] J. Halpern, B. Moszkowski, and Z. Manna. A hardware semantics based on temporal intervals. In J. Diaz, ed., *International Colloquium on Automata, Languages, and Programming (ICALP'83)*, volume 154 of *Lecture Notes in Computer Science*, pages 278–291. Springer-Verlag, 1983.

[HMP92a] T. A. Henzinger, Z. Manna, and A. Pnueli. Timed transition systems. In J. W. de Bakker, K. Huizing, W. P. de Roever, and G. Rozenberg, eds., *Real Time: Theory in Practice*, volume 600 of *Lecture Notes in Computer Science*, pages 226–251. Springer-Verlag, 1992.

[HMP92b] ———. What good are digital clocks? In W. Kuich, ed., *Automata, Languages, and Programming (ICALP'92)*, volume 623 of *Lecture Notes in Computer Science*, pages 545–558. Springer-Verlag, 1992.

[HMP93] ———. Towards refining temporal specifications into hybrid systems. In Grossman et al. [GNRR93], pages 60–76.

[HO93] M. R. Hansen and E.-R. Olderog. Constructing circuits from decidable duration calculus. Unpublished Note, Fachbereich Informatik, Universität Oldenburg, Germany, June 1993.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes.* Series in Computer Science. Prentice-Hall Intl., 1985.

[Hoa90] ——. Refinement algebra proves correctness of compiling specifications. Technical report PRG-TR-6-90, Programming Research Group, Oxford University Computing Laboratory, England, 1990.

[Hoa96] ——. How did software get so reliable without proof? In M.-C. Gaudel and J. Woodcock, eds., *FME'96: Industrial Benefit and Advances in Formal Methods*, volume 1051 of *Lecture Notes in Computer Science*, pages 1–17. Springer-Verlag, 1996.

[HP74] F. J. Hill and G. R. Peterson. *Introduction to Switching Theory and Logical Design.* John Wiley & Sons, 1974.

[HPB93] He Jifeng, I. Page, and J. Bowen. Towards a provably correct hardware implementation of Occam. In G. J. Milne and L. Pierre, eds., *Correct Hardware Design and Verification Methods*, volume 683 of *Lecture Notes in Computer Science*, pages 214–226. Springer-Verlag, 1993.

[HZ95] M. R. Hansen and Zhou Chaochen. Lecture notes on logical foundations of duration calculus. Technical Report ID-TR: 1995-166, Department of Computer Science, Technical University of Denmark, Lyngby, DK, August 1995.

[HZS92] M. R. Hansen, Zhou Chaochen, and J. Staunstrup. A real-time duration semantics for circuits. In *TAU'92: 1992 Workshop on Timing Issues in the Specification and Synthesis of Digital Systems, Princeton University, NJ.* ACM/SIGDA, 1992.

[IEEE78] IEEE standard definitions, symbols, and characterization of metal-nitride-oxide field-effect transistors. IEEE Std. 581, April 1978.

[IEEE88] IEEE standard VHDL language reference manual. IEEE Std. 1076-1987, IEEE Computer Society and IEEE Design Automation TC, 1988.

[Jon86] C. B. Jones. *Systematic Software Development using VDM.* Prentice-Hall Intl., 1986.

[JP96] B. Jonsson and J. Parrow, eds. *Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT '96)*, volume 1135 of *Lecture Notes in Computer Science.* Springer-Verlag, 1996.

[Kle56] S. C. Kleene. Representations of events in nerve nets and finite automata. In C. E. Shannon and J. McCarthy, eds., *Automata Studies*, pages 3–42. Princeton University Press, 1956.

[KM91] R. P. Kurshan and K. L. McMillan. Analysis of digital circuits through symbolic reduction. *IEEE Transactions on Computer-Aided Design*, 10(11):1356–1371, November 1991.

[Lak96] Y. Laknech. *Specification and Verification of Hybrid and Real-Time Systems.* Dissertation, Technische Fakultät der Christian-Albrechts-Universität Kiel, Germany, 1996.

[Lam91] L. Lamport. The temporal logic of actions. Technical report, Digital Systems Research Center, Palo Alto, California, USA, December 1991.

[Lam93] ——. Hybrid systems in TLA$^+$. In Grossman et al. [GNRR93], pages 77–102.

[LdRV94] H. Langmaack, W.-P. de Roever, and J. Vytopil, eds. *Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT '94)*, volume 863 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.

[Lee90] J. v. Leeuwen, ed. *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics. North-Holland, 1990.

[LH94] Y. Lakhneche and J. Hooman. Reasoning about durations in metric temporal logic. In Langmaack et al. [LdRV94], pages 488–510.

[LL95] C. Lewerentz and T. Lindner, eds. *Formal Development of Reactive Systems: Case Study Production Cell*, volume 891 of *Lecture Notes in Computer Science*. Springer-Verlag, January 1995.

[Lue79] J. W. Luenberger. *Introduction to Dynamic Systems. Theory, Models & Applications*. John Wiley & Sons, 1979.

[LZ93] P.-S. Lin and C. Zukowski. Jitter due to signal history in digital logic circuits and its control strategies. In *Proceedings of the 1993 IEEE International Symposium on Circuits and Systems*, volume 3, pages 2114–2117. IEEE, 1993.

[MC80] C. Mead and L. Conway. *Introduction to VLSI Systems*. Addison-Wesley Series in Computer Science. Addison-Wesley Publishing Company Inc., 1980.

[McM93] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[Mil89] R. Milner. *Communication and Concurrency*. Series in Computer Science. Prentice-Hall Intl., 1989.

[Mit90] J. C. Mitchell. Logical relations. In Leeuwen [Lee90], chapter 8.3, pages 415–431.

[ML88] A. Möschwitzer and K. Lunze. *Halbleiterelektronik*. VEB Verlag Technik, eighth edition, 1988.

[MMP+95] O. Matz, A. Miller, A. Potthoff, W. Thomas, and E. Valkema. Report on the program AMoRE. Technical report 9507, Institut für Informatik und Prakt. Mathematik der Christian-Albrechts-Universität Kiel, Germany, 1995.

[Mos85] B. Moszkowski. A temporal logic for multi-level reasoning about hardware. *IEEE Computer*, 18(2):10–19, 1985.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*, volume 1. Springer-Verlag, 1992.

[MP93a] ——. Models for reactivity. *Acta Informatica*, 30(2):609–678, October 1993.

[MP93b] ——. Verifying hybrid systems. In Grossman et al. [GNRR93], pages 4–35.

[MP95] E. W. Meyer and C. Puech, eds. *Symposium on Theoretical Aspects of Computer Science (STACS 95)*, volume 900 of *Lecture Notes in Computer Science*. Springer-Verlag, March 1995.

[MPS95] O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems. In Meyer and Puech [MP95], pages 229–242.

[MS73] A. R. Meyer and L. Stockmeyer. Nonelementary word problems in automata and logic. In *Proc. AMS Symposium on Complexity of Computation*, April 1973.

[NK93] A. NERODE AND W. KOHN. Models for hybrid systems: Automata, topologies, controllability, observability. In Grossman et al. [GNRR93], pages 317–356.

[NOSY93] X. NICOLLIN, A. OLIVERO, J. SIFAKIS, AND S. YOVINE. An approach to the description and analysis of hybrid systems. In Grossman et al. [GNRR93], pages 149–178.

[NT94] S. NADJM-TEHRANI. *Reactive Systems in Physical Environments: Compositional Modelling and Framework for Verification.* Phd thesis, Dept. of Computer and Information Science, Linköping University, March 1994.

[Pan96] P. K. PANDYA. Weak chop inverses and liveness in mean-value calculus. In Jonsson and Parrow [JP96], pages 148–167.

[Pet86] C. A. PETRI. Forgotten topics of net theory. In W. Brauer, W. Reisig, and G. Rozenberg, eds., *Petri Nets: Applications and Relationship to Other Models of Concurrency*, volume 255 of *Lecture Notes in Computer Science*, pages 500–514. Springer-Verlag, 1986.

[PR94] J. L. PETERSEN AND H. RISCHEL. Formalizing requirements and design for a production cell system. In *Symposium ADPM '94: Automatisation des Processes Mixtes: Les Systemes Dynamiques Hybrides.* Belgian Institute of Automatic Control, IBRA, 1994.

[Rav95a] A. P. RAVN. Private communication, February 1995.

[Rav95b] ——. *Design of Embedded Real-Time Computing Systems.* Doctoral dissertation, Department of Computer Science, Danish Technical University, Lyngby, DK, October 1995. Available as technical report ID-TR: 1995-170.

[RDM+93] Y. S. RAMAKRISHNA, L. K. DILLON, L. E. MOSER, P. M. MELLIAR-SMITH, AND G. KUTTY. A real-time interval logic and its decision procedure. In R. K. Shyamasundar, ed., *Foundations of Software Technology and Theoretical Computer Science*, volume 761 of *Lecture Notes in Computer Science.* Springer-Verlag, December 1993.

[RR94] A. W. ROSCOE AND G. M. REED. *Domains for Denotational Semantics.* Series in Computer Science. Prentice-Hall Intl., 1994.

[RRH93] A. P. RAVN, H. RISCHEL, AND K. M. HANSEN. Specifying and verifying requirements of real-time systems. *IEEE Transactions on Software Engineering*, 19(1):41–55, January 1993.

[SB93] P. R. STEPHAN AND R. K. BRAYTON. Physically realizable gate models. Technical Report UCB/ERL M93/33, Electronics Research Laboratory, University of California, Berkeley, May 1993.

[Sei80] C. L. SEITZ. System timing. In *Introduction to VLSI Systems* [MC80], chapter 7.

[Sha49] C. E. SHANNON. The mathematical theory of communication. In *The Mathematical Theory of Communication*, pages 3–91. The University of Illinois Press: Urbana, 1949.

[Sin94] J. SINGH. *Semiconductor Devices.* McGraw-Hill Series in Electrical and Computer Engineering. McGraw-Hill Inc., 1994.

[Ska94] J. U. SKAKKEBÆK. Liveness and fairness in duration calculus. In B. Jonsson and J. Parrow, eds., *CONCUR'94*, volume 836 of *Lecture Notes in Computer Science*, pages 283–298. Springer-Verlag, 1994.

[Sno89] C. M. SNOWDEN, ed. *Semiconductor Device Modelling.* Springer-Verlag, 1989.

[Spi89] J. M. Spivey. *The Z Notation: A Reference manual.* Prentice-Hall Intl., 1989.

[SS94] J. U. Skakkebæk and P. Sestoft. Checking validity of duration calculus formulas. ProCoS Technical Report ID/DTH JUS 3/1, Technical University of Denmark, March 1994.

[SSM+93] E. Sternheim, R. Singh, R. Madhavan, et al. *Digital Design and Synthesis with Verilog HDL.* Design Automation Series. Automata Publishing Company, San Jose,California, 1993.

[Str94] J.-E. Strömberg. *A mode switching modelling philosophy.* PhD thesis, Institutionen för Systemteknik, Universitetet i Linköping, 1994.

[Tho90] W. Thomas. Automata on infinite objects. In Leeuwen [Lee90], chapter 4, pages 133–191.

[Tho95] ——. On the synthesis of strategies in infinite games. In Meyer and Puech [MP95], pages 1–13.

[Ven91] Y. Venema. A modal logic for chopping intervals. *J. Logic of Computation*, 1(4):453–476, 1991.

[VW86] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *1st Symposium on Logic in Computer Science.* IEEE, 1986.

[WE85] N. Weste and K. Eshragian. *Principles of CMOS VLSI Design.* Addison-Wesley VLSI Systems Series. Addison-Wesley Publishing Company Inc., 1985.

[Wil94a] T. Wilke. *Automaten und Logiken zur Beschreibung zeitabhängiger Systeme.* Dissertation, Technische Fakultät der Christian-Albrechts-Universität Kiel, Germany, 1994.

[Wil94b] ——. Specifying timed state sequences in powerful decidable logics and timed automata. In Langmaack et al. [LdRV94], pages 694–715.

[WS63] F. M. Wanlass and C. T. Sah. Nanowatt logic using field-effect metal-oxide-semiconductor triodes. In *Proceedings of the IEEE International Solid-State Circuits Conference, Philadelphia, Pa., 1963*, pages 32–33. IEEE, 1963.

[WTH92] H. Wong-Toi and G. Hoffmann. The control of dense real-time discrete event systems. Technical report STAN-CS-92-1411, Stanford University, 1992.

[ZH96] Zhou Chaochen and M. R. Hansen. Chopping a point. Handout at the 4th ProCoS Working Group Meeting, March 1996.

[Zho93] Zhou Chaochen. Duration calculi: An overview. In D. Bjørner, M. Broy, and I. V. Pottosin, eds., *Formal Methods in Programming and Their Applications*, volume 735 of *Lecture Notes in Computer Science*, pages 256–266. Springer-Verlag, 1993.

[ZHR91] Zhou Chaochen, C. A. R. Hoare, and A. P. Ravn. A calculus of durations. *Information Processing Letters*, 40(5):269–276, 1991.

[ZHRR92] Zhou Chaochen, M. R. Hansen, A. P. Ravn, and H. Rischel. Duration specifications for shared processors. In J. Vytopil, ed., *Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'92)*, volume 571 of *Lecture Notes in Computer Science*, pages 21–32. Springer-Verlag, 1992.

[ZHS93] ZHOU CHAOCHEN, M. R. HANSEN, AND P. SESTOFT. Decidability and undecidability results for duration calculus. In P. Enjalbert, A. Finkel, and K. W. Wagner, eds., *Symposium on Theoretical Aspects of Computer Science (STACS 93)*, volume 665 of *Lecture Notes in Computer Science*, pages 58–68. Springer-Verlag, 1993.

[ZL94] ZHOU CHAOCHEN AND LI XIAOSHAN. A mean value calculus of durations. In A. W. Roscoe, ed., *A Classical Mind — Essays in Honour of C.A.R. Hoare*, chapter 25, pages 431–451. Prentice-Hall Intl., 1994.

[ZRH92] ZHOU CHAOCHEN, A. P. RAVN, AND M. R. HANSEN. An extended duration calculus for hybrid real-time systems. Handout at the 1st Workshop on Theory of Hybrid Systems, Lyngby, Denmark, October 1992.

[ZRH93] ———. An extended duration calculus for hybrid real-time systems. In Grossman et al. [GNRR93], pages 36–59.

[ZZYL94] ZHOU CHAOCHEN, ZHANG JINGZHONG, YANG LU, AND LI XIAOSHAN. Linear duration invariants. In Langmaack et al. [LdRV94], pages 86–109.

Appendix A

Encoding Timed Automata with Stop-Watches in Duration Logic

At first glance, the subset DL of Duration Calculus defined in chapter 5 and used throughout this thesis may seem utterly uninteresting, as it neither allows freeze variables nor comparison of durations (i.e. formulae of the form $\int P \sim \int Q$, where $\sim$ is a relation between real numbers). This section provides a rigorous argument against this impression. By showing that reachability problems of timed transition tables *with stop-watches* are encodable in DL we show that its expressiveness — as far as safety properties are concerned — goes well beyond that of timed automata[1] and that of the considerable number of dense-time logics dedicated towards describing timed automata. As the reachability problem of timed transition tables with stop-watches is undecidable (e.g. [HKPV95, Theorem 4.1]), the construction also yields a proof of undecidability of DL that uses completely different principles than the undecidability results reported in [ZHS93]. Furthermore, this proof generalizes to undecidability of DL on n-bounded models for $n \geq 2$, thus providing an alternative proof of theorem 6.7, and — using results from [Wil94a] on timed automata on 1-bounded models — even for $n = 1$, a problem left open in section 6.2.

A.1 Encoding reachability problems

To prepare for an encoding of their reachability problems in DL, we start with a formal definition of timed transitions tables with stop-watches and their runs.[2]

Definition A.1 (Timed transition table with stop-watches). *A timed transition table $A = (\Sigma, T, C, i, m, r, g, a, t_0)$ consists of*

- *a set Σ of states,*

[1] In the sense of Alur and Dill [AD94], i.e. featuring clocks of slope 1, but neither stop-watches nor other multiple-slope clocks.

[2] Essentially the same devices are called *rectangular automata with stop-watches* in [HKPV95]. We prefer to restrict the use of the term 'automaton' to devices that come equipped with an acceptance condition, and follow the terminology of Alur and Dill [AD94] in calling the corresponding base object without an acceptance condition a *transition table.*

- *a set T of transitions,*
- *a set C of clocks,*
- *an invariant $i : \Sigma \to C \to \mathit{TimeInterval}^+$ associating to each state for each clock an interval that its value must remain in,*
- *a function $m : T \to \Sigma \times \Sigma$ associating to each transition a move within the state set,*
- *a reset function $r : T \to \mathcal{P}(C)$ associating to each transition those clocks that are to be reset upon that transition,*
- *a guard function $g : T \to C \to \mathit{TimeInterval}^+$ associating to each transition the enabling conditions, which are conjunctions of interval constraints on the clock values,*[3]
- *an activity function $a : \Sigma \to \mathcal{P}(C)$ associating to each state those clocks that are currently active, i.e. have slope 1. All other clocks are stopped in that state, i.e. have slope 0, and*
- *an initial transition t_0, selecting the initial state,*[4] *and resetting all clocks, i.e. $r(t_0) = C$.*

Definition A.2 (Runs of a timed transition table). *A* run *of a timed transition table $A = (\Sigma, T, C, i, m, r, g, a, t_0)$ is a (finite or infinite) sequence of pairs $(\sigma_i, v_i) \in \Sigma \times (C \to \mathit{Time})$ satisfying the following constraints:*

- Initiation: $\sigma_0 = m(t_0)\downarrow_2$ *and* $v_0(c) = 0$ *for each* $c \in C$.
- Consecution: *For all indices $i \in \mathbb{N}$ such that $i+1$ belongs to the sequence there is a transition $t \in T$ and a time difference $\delta \in \mathit{Time}$ such that*
  - $(\sigma_i, \sigma_{i+1}) = m(t)$,
  - $v_i^\delta(c) \in g(t)(c)$ *holds for each clock c, where*

    $$v_i^\delta(c) \stackrel{\text{def}}{=} \begin{cases} v_i(c) + \delta & \text{iff } c \in a(\sigma_i), \\ v_i(c) & \text{iff } c \notin a(\sigma_i) \end{cases}$$

    is the reading of c after δ time units spent in state σ_i,
  - *for each clock c,* $v_{i+1}(c) = \begin{cases} 0 & \text{iff } c \in r(t), \\ v_i^\delta(c) & \text{iff } c \notin r(t), \end{cases}$
  - *and* $v_i^{\delta'}(c) \in i(\sigma_i)(c)$ *for each clock c and any* $\delta' \in [0, \delta]$.

[3]Note that disjunctions can be built using more than one transition between the same pair of states.

[4]As transitions are immediate, the concepts of an initial transition and of an initial state coincide. The latter is more common, but we prefer the former as it simplifies the encoding in DL.

A state σ of a timed transition table A is called *reachable* iff there is a run of A that traverses through σ. Reachability of states in timed transition tables with stop-watches is known to be undecidable [HKPV95, Theorem 4.1].

The basic idea of our encoding in DL is to associate to a timed transition table A a DL formula ϕ_A such that the models of ϕ_A are alternating sequences of phases encoding time spent in states and of phases encoding transitions. More precisely, time spent in states is represented on a one-to-one basis, i.e. t time units spent in a state s are represented by a phase of length t where a corresponding state variable $\bar{s}$ is true, whereas transitions are prolonged to also take time such that there is a phase of positive duration that can be used for resetting those clocks that are to be reset during the encoded transition. Clearly, time spent in the artificially prolonged transitions should not be counted by any stop-watch because transitions are immediate in the transition-table model. This and the effect that some stop-watches are stopped in certain states is accomplished by modelling a clock x through a state variable $\bar{x}$ that is true if and only if the encoding is in a phase that models a state where x has slope 1. The actual clock-reading of x thus corresponds to the duration of $\bar{x}$ since the last reset of x. Finally, a reset of x is modelled by setting a state variable r_x to true for a positive duration. This happens within those transition phases that reset x. Figure A.1 on the following page illustrates this simulation of the runs of A by the models of ϕ_A.

Given a timed transition table $A = (\Sigma, T, C, i, m, r, g, a, t_0)$ the detailed encoding in DL proceeds as follows.

1. Every state $\sigma \in \Sigma$ is represented by a state variable $\overline{\sigma}$ of DL, the intuition being that phases where the transition table remains in state σ are represented by a phase of equal length where $\overline{\sigma}$ is true in the trajectory.

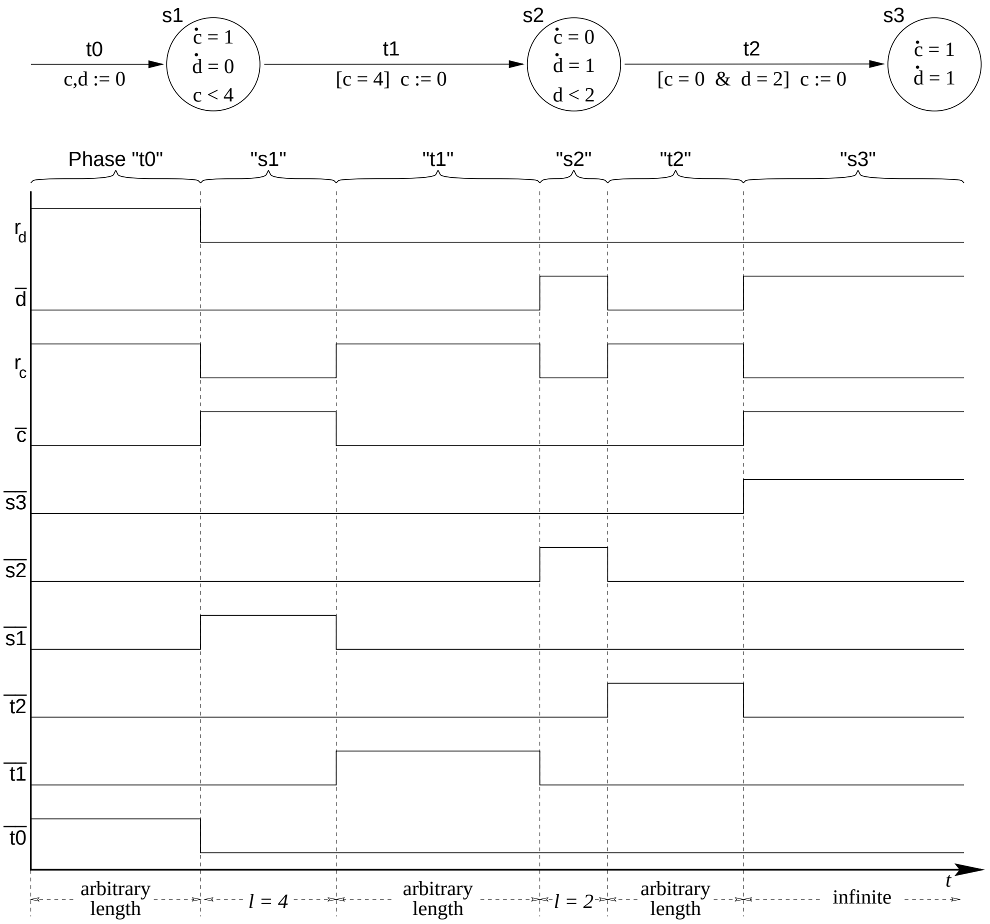
2. Every transition $t \in T$ is represented by a state variable $\overline{t}$ of DL, the intuition being that the transition t being taken by the transition table is represented by a phase of positive duration where $\overline{t}$ is true in the trajectory. We will ensure that this duration is not registered by any clock.

3. The transition table is always in a state or in a transition,
$$\int \left(\bigvee_{v \in \Sigma \cup T} \overline{v} \right) = l \ ,^{5}$$

4. but it is never in two of those at the same time, i.e. for each $v, w \in \Sigma \cup T$ with $v \neq w$
$$\int \overline{v} \wedge \overline{w} = 0 \ .$$

[5] Note that due to finiteness of Σ and T, $\bigvee_{v \in \Sigma \cup T} \overline{v}$ is just an abbreviation that can be syntactically unfolded.

A timed transition table with stop-watches (top of figure)
and a typical trajectory satisfying its encoding in DL (bottom)



The timed trace of the timed transition table is encoded through an alternating sequence of phases encoding transitions and of phases encoding time spent in states. A clock x advancing with slope 1 in a state s is represented by the corresponding state variable $\bar{x}$ being true during the phase corresponding to s, such that the duration of $\bar{x}$ grows with slope 1 during such phases. Correspondingly, a clock x not advancing in state s (i.e. having slope 0) is encoded by $\bar{x}$ being false during the phase corresponding to s.
During phases that encode transitions, all state variables $\bar{x}$ corresponding to clocks x are false, as transitions do not take time in timed transition tables, which is represented by not letting the clock readings grow in the encoding. Reset of a clock x during a transition t is represented through truth of the state variable r_x during the transition phase corresponding to t such that the actual clock reading of x in a time instant is encoded through the accumulated duration of $\bar{x}$ since the last time that r_x was true.

Figure A.1: Encoding timed transition tables in DL

5. Each clock $c \in C$ is modelled by taking the duration over a state variable $\overline{c}$ of DL, where the state variable $\overline{c}$ is true in phases that encode transition-table states where c is active, and false in phases that encode transition-table states where c is stopped. I.e. for each state $\sigma \in \Sigma$ with $c \in a(\sigma)$,

$$\Box\,(\lceil\,\rceil \vee \lceil \overline{\sigma} \Rightarrow \overline{c} \rceil)$$

holds, and for each $\sigma \in \Sigma$ with $c \notin a(\sigma)$,

$$\Box\,(\lceil\,\rceil \vee \lceil \overline{\sigma} \Rightarrow \neg\overline{c} \rceil)$$

holds.

6. In the timed transition-table model, transitions are immediate. Hence, a transition does not affect the readings clocks that it does not reset. When modelling transitions by phases of a positive duration, we have to take care of this by stopping the clocks during transition phases. Hence, for each transition $t \in T$ and each clock c,

$$\Box\,(\lceil\,\rceil \vee \lceil \overline{t} \Rightarrow \neg\overline{c} \rceil) \quad .$$

7. Resetting a clock c is modelled by setting a DL state variable r_c to true. Hence, r_c has to be true exactly in those phases that encode transitions resetting c:
For each $t \in T$ with $c \in r(t)$,

$$\Box\,(\lceil\,\rceil \vee \lceil \overline{t} \Rightarrow r_c \rceil) \quad ,$$

and for each $v \in \Sigma$ as well as for each $v \in T$ with $c \notin r(v)$,

$$\Box\,(\lceil\,\rceil \vee \lceil \overline{v} \Rightarrow \neg r_c \rceil) \quad .$$

8. The transition table starts with the initial transition:

$$\lceil\,\rceil \vee (\lceil \overline{t_0} \rceil\,;\,\mathtt{true})$$

9. States $\sigma \in \Sigma$ are, if left, followed by one of the outgoing transitions. Therefore,

$$\lceil \overline{\sigma} \rceil \leadsto \left(\lceil \overline{\sigma} \rceil \vee \bigvee_{t \in T_\sigma} \lceil \overline{t} \rceil \right) \quad ,$$

where $T_\sigma \stackrel{\text{def}}{=} \{ t \in T \mid \exists\, \sigma' \in \Sigma \,.\, m(t) = (\sigma, \sigma') \}$ is the set of outgoing transitions of σ.

10. Transition phases are in general left towards the target state. But there is a subtlety here: in the timed transition-table model, states can be left immediately after entering such that a second transition may follow the first instantaneously. Therefore, for any transition $t \in T$ with $m(t) = (\sigma, \sigma')$,

$$\lceil \overline{t} \rceil \leadsto \left(\lceil \overline{t} \rceil \vee \lceil \overline{\sigma'} \rceil \vee \bigvee_{t' \in T_{\sigma'}} \lceil \overline{t'} \rceil \right) \quad .$$

11. Constraints on clock values have to be encoded. As the encoding is such that the current reading of clock c corresponds to the duration of $\overline{c}$ since the last occurrence of r_c, violation of a clock constraint $c \in I$, where c is a clock and I is an interval, can be represented by the DL formula

$$\overline{c \notin I} \stackrel{\text{def}}{=} \texttt{true}\,;\,\lceil r_c \rceil\,;\,(\lceil \neg r_c \rceil \wedge \textstyle\int c \notin I) \quad .$$

Here, $\int c \notin I$ abbreviates the following predicates depending on the shape of the interval I:

$$\begin{aligned} \textstyle\int c \notin [a,b] &\;\hat{=}\; \textstyle\int c < a \vee \int c > b \ , \\ \textstyle\int c \notin (a,b] &\;\hat{=}\; \textstyle\int c \leq a \vee \int c > b \ , \\ \textstyle\int c \notin [a,b) &\;\hat{=}\; \textstyle\int c < a \vee \int c \geq b \ , \\ \textstyle\int c \notin (a,b) &\;\hat{=}\; \textstyle\int c \leq a \vee \int c \geq b \ . \end{aligned}$$

Whenever the observation interval covers (at least) the last reset point of c, predicate $\overline{c \notin I}$ holds iff the clock reading of c in the endpoint of the observation interval is outside I. The predicate fails directly if the last reset point is not within the observation interval and thus the actual clock reading cannot be determined.

12. Using this, the invariant i can be encoded as

$$\bigwedge_{\sigma \in \Sigma, c \in C} \Big(\texttt{true}\,;\,\lceil \overline{\sigma} \rceil \Rightarrow \neg\, \overline{c \notin i(\sigma)(c)} \Big) \quad .$$

13. The guards of transitions can be simulated by a similar construction. A transition $t \in T$ is never entered if one of its guarding conditions is violated:

$$\left(\texttt{true}\,;\,\lceil \neg \overline{t} \rceil \wedge \bigvee_{c \in C} \overline{c \notin g(t)(c)} \right) \rightsquigarrow \lceil \neg \overline{t} \rceil$$

For each timed transition table A, let ϕ_A be the conjunction of the formulae constructed according to rules 1 to 13.

Let A be a timed transition table (possibly with stop-watches) and let σ be a state of A. The construction of ϕ_A is such that

Proposition A.3. *σ is reachable in A iff there is a trajectory tr and $e \in Time$ such that $tr, [0, e] \models \phi_A \wedge (\texttt{true}\,;\,\lceil \overline{\sigma} \rceil)$.*

By definition of satisfaction in DL, $tr, [0, e] \models \phi_A \wedge (\texttt{true}\,;\,\lceil \overline{\sigma} \rceil)$ implies that tr does not satisfy $\neg\,(\phi_A \wedge (\texttt{true}\,;\,\lceil \overline{\sigma} \rceil))$. Thus, an immediate consequence of proposition A.3 is that validity of that formula corresponds to reachability of σ in A:

Corollary A.4 (Correspondence between reachability and validity). *A given state σ is reachable in A iff there is a trajectory $tr \in Traj$ such that $tr \not\models \neg\,(\phi_A \wedge (\texttt{true}\,;\,\lceil \overline{\sigma} \rceil))$. I.e., σ is reachable in A iff $\neg\,(\phi_A \wedge (\texttt{true}\,;\,\lceil \overline{\sigma} \rceil))$ is invalid.* ▟

Thus, reachability problems in timed transition tables with stop-watches correspond to validity problems of certain DL formulae. As timed transition tables with stop-watches are considerably more expressive than timed automata, which are a widely studied formalism for describing real-time systems, this shows that DL is indeed quite expressive. Another consequence of the correspondence is that undecidability results for timed transition tables with stop-watches carry over to DL.

A.2 Undecidability results

As the reachability problem for timed transitions tables with stop-watches is undecidable [HKPV95, Theorem 4.1], undecidability of validity of DL formulae follows from encodability of those reachability problems in DL.

Theorem A.5 (Undecidability of DL on finitely variable models). *Validity of formulae of Duration Logic on finitely variable models is in general undecidable.*

Proof. Immediately from undecidability of reachability for timed transition tables with stop-watches and the correspondence of those reachability problems to validity of certain DL formula, as stated in corollary A.4. ∎

Furthermore, careful inspection of the formula $\phi_A \wedge (\texttt{true}\,; \lceil \overline{\sigma} \rceil)$ reveals that it is satisfiable by a 2-bounded trajectory if it is satisfiable at all. This is due to the fact that timing constraints are only imposed on the duration of phases encoding states (in clauses 12 and 13 of the definition of ϕ_A), such that transition phases can have arbitrary length. As furthermore discontinuities only occur upon entering and exiting a phase, a 2-bounded (and thus also n-bounded for each $n \geq 2$) trajectory satisfying $\phi_A \wedge (\texttt{true}\,; \lceil \overline{\sigma} \rceil)$ exists if the formula is satisfiable at all. Hence, the following variant of theorem A.5 is also true.

Theorem A.6 (Undecidability of DL on n-bounded models for $n \geq 2$). *Validity of DL formulae on n-bounded models is in general undecidable for each $n \geq 2$.* ∎

Finally, the results of T. Wilke in [Wil94a] on decidability of the language emptiness problem for timed automata with multiple-slope clocks shed light on decidability of DL on 1-bounded models, the case left open in section 6.2:

Theorem A.7 (Undecidability of DL on 1-bounded models). *Validity of DL formulae on 1-bounded models is in general undecidable.*

Proof. In [Wil94a, Theorem 4.7.10], T. Wilke shows that it is undecidable whether a timed automaton with two-slope clocks with two given non-negative slopes accepts any 1-bounded word. The proof of this theorem immediately generalizes from language emptiness to reachability of states. Hence, it is in general undecidable whether a timed automaton with two-slope clocks with two given non-negative slopes (in particular, an automaton with stop-watches) can reach a given state on some 1-bounded run. Through above encoding of reachability problems of timed transition tables with stop watches in DL, this implies undecidability of the validity problem of DL formulae on 1-bounded models. ∎

A.3 Discussion

On one hand, the encoding of reachability problems of timed transition tables with stop watches demonstrates that Duration Logic is indeed quite expressive. On the other hand, it provides a novel proof of undecidability of DL, such that the insights from the proof sketches given in [ZHS93] are consolidated.

Furthermore, our encoding is more stable against changes of the model class than the encoding of the halting problem for two-counter machines given in [ZHS93]: as indicated in section 6.1, the latter encoding crucially depends on unbounded variability of the trajectories. In contrast, the new encoding can be used to derive undecidability results for the validity problem of DL formulae on n-bounded models also (cf. theorems A.6 and A.7), thereby providing an alternative proof and a generalization of theorem 6.7.

Index

Entries are classified as being either a defining occurrence of the term, or a formal result (i.e., theorem, lemma, corollary, etc.) dealing with the term, or an illustration, or a simple textual occurrence of the term. Typographic conventions are used to distinguish between the different entry classes. Usual textual occurrences are typeset in roman (e.g. 123), defining occurrences are printed boldface (**123**), formal results are italicized (*123*), and entries dealing with illustrations are underlined (<u>123</u>).